FlexiSketch: a lightweight sketching and metamodeling approach for end-users

Wüest, Dustin ; Seyff, Norbert ; Glinz, Martin

Abstract: Engineers commonly use paper and whiteboards to sketch and discuss ideas in early phases of requirements elicitation and software modeling. These physical media foster creativity because they are quick to use and do not restrict in any way the form in which content can be drawn. If the sketched information needs to be reused later on, however, engineers have to spend extra effort for preserving the information in a form that can be processed by a software modeling tool. While saving information in a machine-readable way comes for free with formal software modeling tools, they typically anticipate the use of specific, predefined modeling languages and therefore hamper creativity. To combine the advantages of informal and formal tools, we have developed a flexible tool-supported modeling approach that augments a sketching environment with lightweight metamodeling capabilities. Users can create their own modeling languages by defining sketched constructs on demand and export model sketches as semiformal models. In this article, we first give an overview of FlexiSketch and then focus on an evaluation of our approach with two studies conducted with both novice modelers and experienced practitioners. Our goal was to find out how well modelers manage to use our lightweight metamodeling mechanisms, and how they build notations collaboratively. Results show that experienced modelers adopt our approach quickly, while novices have difficulties to distinguish between the model and metamodel levels and would benefit from additional guidance and user awareness features. The lessons learned from our studies can serve as advice for similar flexible modeling approaches.

DOI: https://doi.org/10.1007/s10270-017-0623-8

FlexiSketch: a lightweight sketching and metamodeling approach for end-users

Dustin Wüest[1,2] · **Norbert Seyff**[1,2] · **Martin Glinz**[1]

Abstract Engineers commonly use paper and whiteboards to sketch and discuss ideas in early phases of requirements elicitation and software modeling. These physical media foster creativity because they are quick to use and do not restrict in any way the form in which content can be drawn. If the sketched information needs to be reused later on, however, engineers have to spend extra effort for preserving the information in a form that can be processed by a software modeling tool. While saving information in a machine-readable way comes for free with formal software modeling tools, they typically anticipate the use of specific, predefined modeling languages and therefore hamper creativity. To combine the advantages of informal and formal tools, we have developed a flexible tool-supported modeling approach that augments a sketching environment with lightweight metamodeling capabilities. Users can create their own modeling languages by defining sketched constructs on demand and export model sketches as semiformal models. In this article, we first give an overview of FlexiSketch and then focus on an evaluation of our approach with two studies conducted with both novice modelers and experienced practitioners. Our goal was to find out how well modelers manage to use our lightweight metamodeling mechanisms, and how they build notations collaboratively. Results show that experienced modelers adopt our approach quickly, while novices have difficulties to distinguish between the model and metamodel levels and would benefit from additional guidance and user awareness features. The lessons learned from our studies can serve as advice for similar flexible modeling approaches.

Keywords Requirements engineering · Tool · Sketching · Ad hoc modeling · Notation definition · End-user metamodeling · Lightweight metamodeling · Collaborative metamodeling · Evaluation

1 Introduction

Despite all technological advances, physical media such as whiteboards, flip charts, and paper still play an important role in software projects [4,15,37,55]. Engineers particularly use them to sketch and discuss new ideas. These creative activities may happen anytime in a project, but are especially important for early project phases where people discuss requirements or early solution ideas [4,17,38]. In a creative process, engineers need to sketch ideas in any form and on different levels of detail, sometimes using elements of a modeling language, sometimes inventing notations on the fly [39]. Frequently, notations will be chosen such that involved business stakeholders (e.g., customers, domain experts) can understand them without explicit training [11]. Ambiguity is accepted as an important characteristic of creativity and idea generation [17]. However, the resulting sketches are difficult to reuse and to integrate into the documentation of the emerging system [2]. Documenting them as uninterpreted photographs hampers later reuse and understanding, as the interpretations that the creators had in mind are lost. Recre-

✉ Dustin Wüest
dustin.wueest@fhnw.ch

Norbert Seyff
norbert.seyff@fhnw.ch

Martin Glinz
glinz@ifi.uzh.ch

[1] Department of Informatics, University of Zurich, Zurich, Switzerland

[2] FHNW, University of Applied Sciences and Arts Northwestern Switzerland, Windisch, Switzerland

Software and Systems Modeling (Springer)
DOI 10.1007/s10270-017-0623-8 Published online 16 September 2017
Preprint in final layout
Final publication is available from the publisher at: https://link.springer.com/article/10.1007/s10270-017-0623-8
A full-text view only version is provided by Springer Nature SharedIt: http://rdcu.be/vVzx

ating the sketched information as models in an established modeling language could preserve intended meanings, but is a tedious and error-prone manual translation process [4,42]. Further, as more time passes before this translation happens, it gets more difficult to remember the original intentions and to perform an accurate translation. Therefore, there is a need for method and tool support for creating sketches on suitable media and documenting them such that the sketched information can be reused and its interpretation is preserved [42]. This need becomes even more urgent with the increasing popularity of model-driven engineering, a software development methodology in which models play a central role in the engineering process.

Although many sketches represent diagrams in some modeling language or another, classic software modeling tools are not suited for supporting this kind of creative early modeling. This is due to the fact that modeling tools restrict modelers to the use of a predefined modeling language with strict rules for syntax and semantics [44]. It enables the tools to provide comprehensive support for creating and verifying models, leading to precise documentation with little or no ambiguity (or even enabling the automatic generation of source code). However, this advantage comes at the expense of not allowing engineers to create free-form sketches or models not adhering to the predefined language syntax. These restrictions stifle creativity [4] and hinder the flow of thoughts.

In our previous work, we have proposed an approach and developed an associated tool [59–61], which aim at supporting creative sketching without the disadvantages of cumbersome reuse and documentation of the sketched information as mentioned above. Our tool provides a sketching interface and enables users to perform lightweight metamodeling by annotating the elements they have sketched with meanings. A simple metamodel gets created semiautomatically on the fly by processing this information and by inferring cardinality rules. As a result, each sketch is stored together with a simple metamodel. How accurate and detailed this metamodel is depends (to a certain degree) on the amount of user annotations. Users are free to decide how much metamodeling they want to perform, according to whether and how they want to reuse the sketches. Thus, our approach provides the flexibility of paper or whiteboards, but at the same time facilitates the integration of model sketches into the overall system design process.

Our tool comes in two versions: FlexiSketch uses tablets as sketching media and supports inexpensive, mobile sketching at any time and in any place. FlexiSketch Desktop runs on electronic whiteboards and can provide a big screen for co-located meetings. Multiple tablets can be connected to the desktop version over Wi-Fi, which allows users to collaborate and simultaneously work in the same workspace with multiple screens.

In this article, we focus on two studies we conducted to evaluate our approach. Central to our approach is the interweaving of sketching and lightweight metamodeling activities, which was also the focus of the studies. Our main goal was to see how well modelers can use the metamodeling features of our approach, and how they define modeling languages in a collaborative setting. In contrast, measuring the quality of the created modeling languages or the generated metamodels was not within the scope of the studies.[1] We particularly answer the following two research questions:

RQ1: *What patterns of sketching and language definition emerge when modelers collaboratively define lightweight modeling languages with our approach?*

We were interested to find out how small groups behave when they define a lightweight modeling language. RQ1 includes the following sub-questions: are there recurrent patterns between the groups? Are there moments of simultaneous sketching? How many group members define a part of the modeling language? When do they perform these definitions? To answer these questions, we performed a study consisting of simulated workshops with small groups of students (novice modelers) and practitioners (expert modelers).

RQ2: *To what extent can novice and expert modelers define lightweight modeling languages correctly and completely with our approach?*

We were particularly interested to find out whether novice modelers manage to define all of their model constructs with our approach, and if their definitions make sense. In other words, if a metamodeling expert would define the same model constructs (limited to the definitions that are possible to create with our approach), and we would take these definitions as ground truth, how well would the solutions from the novice modelers match this ground truth? If the solutions would match the ground truth reasonably well, this would mean that our approach allows users to actually create lightweight metamodels without the help of metamodeling experts. To answer this question, we performed a study consisting of a quantitative experiment with more than 100 students in computing. We complemented the study with a qualitative experiment with experienced practitioners, in order to see whether they find our approach useful for the kinds of customized modeling languages that they use in practice.

This article is an extension of an existing conference paper [62]. The conference paper reports on the simulated workshops and discusses RQ1 for small groups of students and

[1] Since in our case the motivation for performing ("just enough") metamodeling is to formalize sketches rather than defining high-quality modeling languages, it does not make sense to compare the quality of the created metamodels with those built by metamodeling experts at this stage. However, a future study about metamodel quality could help to improve the generated metamodels and thus enhance the reusability of FlexiSketch artifacts.

practitioners. The main new contribution of this extension is a second study consisting of a quantitative experiment with students and a qualitative experiment with practitioners to answer RQ2. Further contributions include (i) a set of guidelines for flexible modeling approaches that we derived from the findings of our two studies and (ii) a comparison of FlexiSketch with other flexible modeling tools.

The remainder of this article is structured as follows. Section 2 describes our tool-supported approach. Section 3 reports on the first study consisting of the simulated workshops for answering RQ1. Section 4 reports on the second study consisting of the quantitative and qualitative experiments for answering RQ2. Section 5 summarizes and discusses the findings from both studies. Section 6 presents an overview of related work. Section 7 provides conclusions and future work.

2 A tool-supported approach for flexible modeling

In this section, we summarize the FlexiSketch approach, present our tool, and briefly describe the intended usage scenarios.

2.1 Core ideas

The main ideas of FlexiSketch are the representation of sketches as node-and-edge diagrams[2] and the interleaving of modeling activities (i.e., drawing elements) and lightweight metamodeling (by annotating drawn elements) [59,60]. This means that FlexiSketch users can perform modeling and metamodeling activities in any order. For example, they can first draw a model and then annotate it, or alternate between modeling and annotating, or create a metamodel for a domain-specific modeling language (DSML) first and then draw a model using this DSML. A sketch in FlexiSketch consists of a set of individual elements, distinguished as symbols (nodes) and links (edges). This provides a basic structure upon which the metamodeling capabilities of FlexiSketch build. Metamodeling happens by assigning types to nodes and links and defining link cardinalities.

2.2 Sketching and editing

Figure 1 shows screenshots of the mobile and desktop versions of FlexiSketch.

The mobile version of FlexiSketch runs on Android devices. After startup, the tool presents a white drawing canvas, inviting for free-form sketching. Upon lifting the finger from the screen for a certain amount of time, FlexiSketch creates an individual object on the sketch canvas that contains the drawn lines: a symbol (node). When the user draws a line from one symbol to another, the tool converts the line into a link between the symbols. A symbol can be user-drawn, an imported shape, or an image. Additionally, the user can add text boxes to elements.

Symbols and links can be selected by tapping, whereupon they are highlighted in blue and reveal a context menu. A highlighted element can be dragged around on the canvas. The context menu allows users to manipulate elements in different ways, e.g., add text, scale, delete, and merge two symbols into one. The context menu of a link also allows a change of its appearance and a choice between a unidirectional and bidirectional link.

A folding menu on the right edge of the screen contains typical editor options like choosing stroke color and thickness. It also has options to add free-floating text boxes, existing images from the filesystem or camera, and provides six standard geometrical shapes including a square, a circle, and a stickman.

More details about the technical solution regarding sketching and a first usability study are presented in [59].

2.3 Lightweight metamodeling and export

FlexiSketch uses a *lightweight metamodeling* mechanism, supporting the definition of types for symbols and links and the definition of cardinalities for link types. Advanced metamodeling concepts such as inheritance, hierarchies, or complex constraints are not supported. This is a deliberate decision, as FlexiSketch is not intended to support metamodeling experts in creating full-fledged modeling languages. Instead, our approach is intended for users with little or no metamodeling knowledge and should provide "just enough" metamodeling features that do not overwhelm those users. Also, users should not get a strong feeling that they are actually metamodeling; as much as possible should happen behind the scenes and should be transparent to the users. Our vision is to support the creation of a coarse metamodel which helps to add meaning to model sketches and also allows for their export/import, so that the sketched content can be refined with other, more formal modeling tools. Currently, the added meaning primarily exists in the type names (assuming that the users have a definition for the respective names in mind) and implicitly in the restrictions defined by the cardinality rules. In other words, we currently focus on storing the concrete and abstract syntax (the metamodel). The semantics is only stored implicitly in the form of names.

To assign types in FlexiSketch, the users first select an element (a symbol or link) and then tap on the *plus* icon in the context menu (see Fig. 1). The appearing dialog then allows them to define and assign a type.

[2] Results of an earlier study have shown that node-and-edge diagrams are among the most frequently used diagram types in early requirements elicitation and design sessions [59].

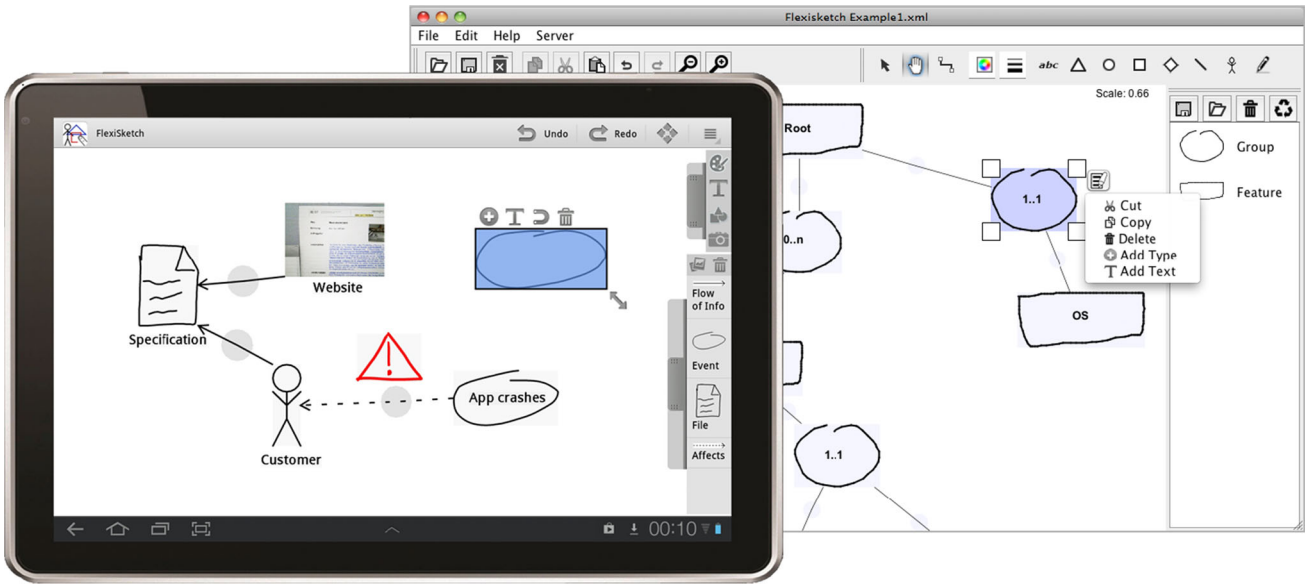

Fig. 1 Screenshots of the mobile and desktop versions of FlexiSketch showing the UIs and some model sketches

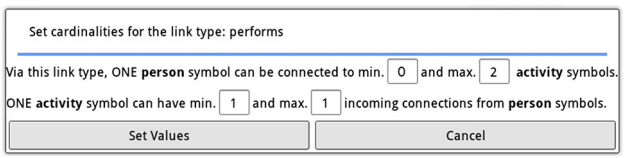

Fig. 2 The cardinality dialog

All type definitions are managed in a type library. This type library can be shown by unfolding a container on the right edge of the screen. The type library provides a drag and drop mechanism that allows users to create new instances of types on the drawing canvas (e.g., a user sketching a sequence of activities could draw and define an activity symbol once and then get copies of the activity symbol via the drag and drop mechanism of the type library).

A *relationship type* is defined by a link type and the two connected symbol types. The user can define cardinalities of a relationship type by selecting a link on the drawing canvas (that serves as an example for that relationship type) and using the corresponding context menu option to open the cardinality dialog (see Fig. 2). There, minimum and maximum cardinalities can be defined for both directions. As a precondition for defining cardinalities of a relationship type, all elements of that relationship type must already have a type assigned (i.e., the link and the two connected symbols).

FlexiSketch includes a sketch recognition algorithm to detect similar symbols: when the user draws a symbol on the sketch canvas, the tool compares that symbol with the type definitions contained in the type library. If it detects potential matches, the tool displays up to three type proposals at the bottom of the screen. The user can either ignore a proposal or tap on it to accept it. In the latter case, the tool assigns that type to the symbol and (depending on the user settings) replaces the symbol with the one defined in the type library.

The same type can be assigned to different looking symbols, which allows the user to have different graphical representations for a single type in the type library. Also, different types can be assigned to similar looking symbols. (The tool does not prevent syntactic ambiguity). In the current tool version, the type library shows only the initially drawn symbol as graphical representation for each type (while the sketch recognizer compares new drawings with all representations). This representation is also used for the drag and drop mechanism.[3] For links, the tool provides a finite set of graphical appearances (currently six) via the context menu of a link. When the user assigns a type to a link and later creates a second link with the same appearance, the tool will automatically assign the same type to the second link.

The main menu of FlexiSketch includes a wizard that checks whether all elements on the drawing canvas have types assigned. When the wizard is launched, it goes through all undefined elements step by step, highlighting and centering each element while asking for a type definition. The user can skip definitions or delete unwanted elements. In the last step, the wizard asks the user to define missing cardinalities. A more comprehensive description of the metamodeling mechanisms of FlexiSketch can be found in [60].

[3] In the next tool version, the user will be able to see a list of all graphical representations for a type and to manage them. This will allow the user to delete a representation, to choose which representation should be used for the drag and drop mechanism, and to replace all symbols with a particular type on the sketch canvas with one of the representations in the list.

Our tool exports two XML files: one contains the model sketch, and the other contains the metamodel. If there is no metamodel information, the sketch XML file can still be used on its own, as it contains a structured list of the sketched elements (describing a generic graph consisting of nodes and edges). If there is a metamodel XML file, the entities from the sketch XML file link to it accordingly.

Furthermore, metamodels can be stored and exported independently from sketches. This allows the user to create and store simple modeling languages that can be reused later on. Thus, there are two ways in which our tool can be used: (i) the user starts to sketch without a metamodel, draws arbitrary node-and-edge diagrams, and adds metamodel information if/when needed, or (ii) the user loads the metamodel of a previously defined modeling language and reuses that language when creating new sketches—while still having the option to augment the language with new constructs on the fly.

The user has the option of converting the XML files to the GOPPRR format, which is used by the commercially available metamodeling tool MetaEdit+ [28]. The sketched model and the metamodel can be imported in MetaEdit+. It is then possible to further refine and augment both artifacts. MetaEdit+ provides a palette with the symbol and link types that the user defined in FlexiSketch. MetaEdit+ also checks the defined cardinality constraints when the user continues to work on the model. Figure 3 shows a model sketch in FlexiSketch and its corresponding exported version in MetaEdit+, including its metamodel.

2.4 Collaboration

2.4.1 Design issues

For adding a collaboration mode to our tool, we identified five key design issues (D). These issues also reflect selected design guidelines that can be found in research about computer supported collaborative work [19–21,54]:

D1: *All meeting participants should be able to edit the workspace simultaneously*. This fosters participation and can save time when work is performed in parallel [54]. The guideline also implies that access to the workspace must be available at more than one single physical location, because restricted access (e.g., due to the limited physical space in front of the input device) can stifle participation [21].

D2: *A tool should prevent conflicting inputs from multiple participants*. This is a sub-issue of supporting the coordination between participants [20].

D3: *A tool should provide both shared and private views*. While a shared view helps to gather the foci of participants [19], private views allow users to create private notes [21].

D4: *All participants should immediately receive the results of a design session*. Since a meeting is an event embedded in a larger work context [21], it must be easy for all participants to take the meeting results with them for later reuse.

D5: *The tool should increase the awareness of each other's actions*. Participants should always be able to know what the other participants are currently doing [20]. The tool should actively support this in situations where the results of user actions cannot be seen within few seconds.

Due to time constraints, we prioritized the design issues and addressed D1 to D4, while D5 is left for future work (although it is partly addressed by the visualization of the locking mechanism—as described below). The resulting tool solution was sufficient for conducting our studies.

2.4.2 Implementation

FlexiSketch Desktop is a version of our tool suited for PCs and electronic whiteboards. The interface looks slightly different because it is adjusted for mouse and keyboard input (but everything is also accessible on touch screens by simple finger touches). However, its functionality is essentially identical to the mobile version.

Our tool supports multi-screen collaboration by using the desktop version as a server and connecting multiple tablets to it over Wi-Fi.[4] Multiple persons can simultaneously work on the same sketch and together define a simple custom modeling language. All changes get synchronized immediately between the connected devices. While different parts of the sketch can be edited concurrently, a locking mechanism ensures that each part is only edited by one person at a time: as soon as a sketched element gets selected, this element becomes locked and appears with a red background for all other users. This mechanism prevents inconsistent states of individual elements (e.g., it cannot happen that one user deletes an element, while another user is in the middle of assigning a type or adding text to the same element). Furthermore, highlighting locked parts of a sketch provides some user awareness, because a user can see the parts of a sketch that are currently being edited by other users. There is no specific indicator telling that another user is currently looking at a popup dialog for changing an element, and the change gets synchronized once the user closes the dialog. However, the element is locked and highlighted during that time, which indicates that a user is changing it in some way. Apart from the locking mechanism, the tool does not support the resolution of conflicts that occur through concurrent inputs. For example, when two users assign the same type to two different symbols, the earlier definition will appear as an entry in the type library, while the second one is silently added to the same entry (enlarging the training set of the sketch recognizer), but not shown to the user. If two users

[4] A demo video of the FlexiSketch tool is available at http://youtu.be/0kHjNfHLViM.

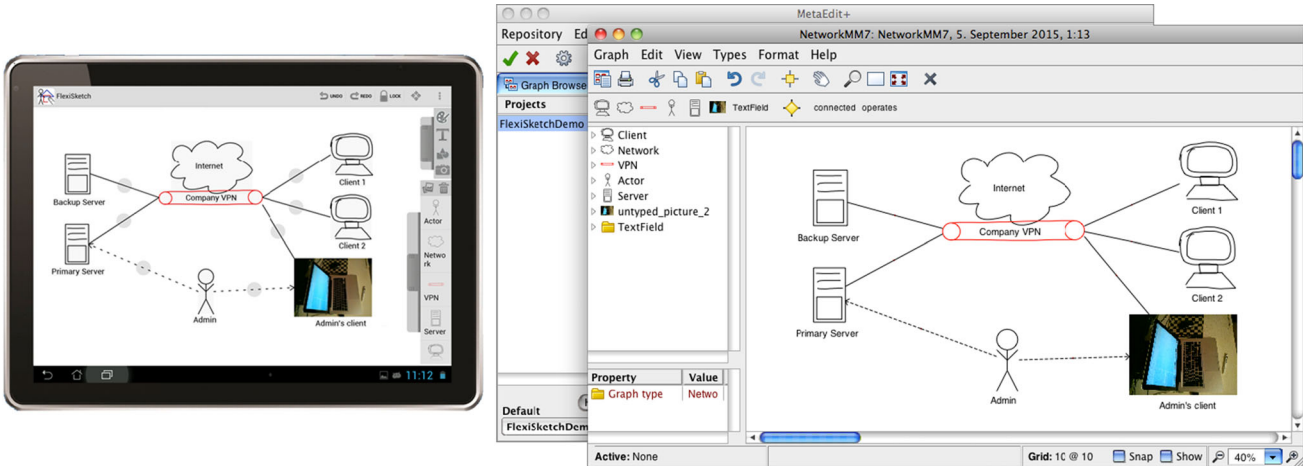

Fig. 3 A model sketch and its metamodel exported to MetaEdit+

assign different types to similarly looking symbols, it is up to them to resolve the introduced ambiguity by deleting one of the type definitions (unless they introduced the ambiguity on purpose).

While the desktop version is not receiving any user input, it shows an overview of the whole sketch canvas and type definitions and can optionally be projected onto a wall. Users can zoom and scroll their individual views on the tablets. A session moderator could steer discussions by using the desktop version to zoom in on certain parts of a sketch or to highlight some parts of it. Due to a technical limitation, the desktop version is currently read-only when it is in collaborative mode, and tablets connected over Wi-Fi must be used as input devices. We plan to change this in a future release of the tool.

A share function allows any user to push their workspace state to all other connected devices or to pull the server state. This allows users to join a session at any time, receiving the current workspace state. Or, a user could disconnect at any time to have a private workspace and reconnect again to share her work.

2.5 Target audience and field of application

The target users of FlexiSketch are: (i) software or systems engineers who create sketches during a system development project and (ii) requirements engineers (business analysts) who use sketches to create and communicate requirements. We expect that the modeling skills as well as metamodeling knowledge of our target audience vary significantly. We assume that FlexiSketch users may or may not have knowledge about or previous experience with metamodeling.

As FlexiSketch has been designed as an alternative to using paper and pencil or whiteboards, it aims at being applied in all situations where creativity and the generation and communication of ideas are central activities, with commensurate model sketches being created in the process. Typical settings are meetings, workshops, and discussions with or among stakeholders where they create ideas, elicit requirements, and create early design solutions. We especially focus on early requirements engineering (RE) sessions. We believe that this is where informal sketches are most frequent [4,17]: early in the software process and when external stakeholders such as customers are present (who might not know standard modeling languages such as UML). As FlexiSketch runs on mobile devices, it can be used at any time and in any place where ideas come to one's mind. This also includes requirements elicitation in situ, i.e., when a stakeholder sketches ideas about requirements in the actual work environment.

3 Study 1: what patterns emerge when modelers collaboratively define modeling languages?

To our knowledge, FlexiSketch is the first software tool that enables the definition of modeling languages collaboratively in a sketching environment. Therefore, with RQ1 we want to investigate how potential users of our approach collaborate during this activity. We performed a qualitative study consisting of simulated workshops with six small student and practitioner groups. Results of this study are useful for both (i) improving our own approach and (ii) future work in the field of collaborative, tool-supported metamodeling. Related work about this topic is still scarce. The traditional view on metamodeling is that a single expert creates a new modeling language beforehand, whereupon modelers then start using it [30]. In contrast, with our approach, modelers can create modeling languages on the fly while sketching models. This allows them to come up with languages that

are suitable for their particular situations (i.e., for creative, early requirements elicitation meetings). They can change or augment existing languages on demand. Because sketching and metamodeling are intertwined in our approach, we also need to analyze how groups sketch together in order to put their metamodeling behavior in the right context.

We chose both practitioners and students to have a mix between more and less experienced engineers. Eight Master students who attended an advanced requirements engineering course at the University of Zurich participated in the study, as well as nine software and RE practitioners from different companies. Some of the students already worked in industry for several years. The simulated workshops were part of the course, but we told the students that this session's purpose is to evaluate our approach and does not affect their course grades. We divided the students into three groups of two or three people, SG1 (S1 and S2), SG2 (S3–S5), and SG3 (S6–S8). We believe this to be a realistic group size for ad hoc meetings where participants come up with new ideas and create model sketches. Similarly, we had three practitioner groups, PG1 (P1–P3), PG2 (P4–P6), and PG3 (P7–P9). PG1 consisted of practitioners from different Swiss companies who knew each other from their time at the university. All of them have similar roles in their respective companies. PG2 consisted of members who work in an Austrian university but regularly receive tasks from industrial partners. P4 is not the direct boss of P5 and P6, but is one level above them in the hierarchy of the university. Members of PG3 work together within an Austrian company focused on mobile applications. P8 is the boss of P7 and P9.

3.1 Method

We could perform this study by using either only the desktop version of our tool on an electronic whiteboard or the mobile version in a multi-screen setup. Since the mobile version stands for the core of our approach, and electronic whiteboards are not available everywhere, we decided to perform the study with the collaborative mobile tool and using PCs or projectors to display a shared overview of the workspace. Every workshop participant received an Android tablet with a screen size between 9.4 and 10.1 inches. The tablets were not identical, but we believe that the same is true in a real-world scenario where practitioners bring their own tablets. The practitioner workshops were conducted in German. Quotations from German speaking participants presented in this article were translated to English.

Every group sat around a table. For student groups, we placed a big screen showing the shared overview on every table. For practitioner groups, we were able to use existing projectors on-site. Figure 4 shows one of the practitioner groups during the study.

Fig. 4 A group of practitioners is working in one of our simulated workshops

First, we gave a 5 min introduction to FlexiSketch and every participant could try the tool in single-user mode for an additional 5 min. We then introduced the collaboration features and connected the tablets with the server. At that point, the main modeling task started. (The task description[5] follows below). We told all groups that they have to solve the modeling task in a collaborative way (but we did not say how; neither did we put restrictions on the seating, nor did we introduce a workshop moderator). A part of the task description stated that all elements of the sketch must have a type assigned at the end of the session. For 20 min (the time was controlled), the groups performed modeling and created type definitions for their modeling languages. Due to a technical limitation, cardinality rules were only inferred automatically. For the manual definition of cardinality rules, we refer to our second study in Sect. 4.

The practitioner groups were asked to think about and choose a current RE-related task or problem from their company,[6] and they were free to choose or invent any modeling language. In order to have a similar initial situation for the student groups, we provided the students with a predefined task, because otherwise (unlike the practitioners) students would not have had a shared task/problem to work on. We also predefined the diagram types (but not the notations) they should use. The practitioner groups would most likely not think of completely new languages to model the selected RE tasks in our simulated workshops, but use something similar to what they have already used before in their daily work life. (This is confirmed by our study results). We concluded that we can get a comparable effect in the student groups by telling them to use diagram types to which they got briefly introduced in their previous studies, while (as novice modelers) being

[5] The handouts and survey for the student groups are available at https://files.ifi.uzh.ch/rerg/flexisketch/StudentHandouts.pdf.

[6] The members of PG1 picked a task from one of their companies and turned it into a more general problem to which all group members could relate to.

still far from becoming experts in understanding and using those diagram types. The students received two modeling tasks about a fictive online learning platform, each lasted for 10 min. They had to draw a use case (UC) diagram, followed by a graphical user interface (GUI)[7] for the use case "sign up on the online portal." The tasks were given in written form (in natural language) and also stated that students should be creative and add additional ideas if possible. Each group was supervised by one of the FlexiSketch creators who did not intervene unless there was a technical problem with the tool.

All sessions concluded with a semi-structured interview with the whole group. Because time during the course was limited, students were further asked to fill out an online survey. This also allowed each student to give individual and anonymous feedback. Seven students completed the survey.

Each group was video-recorded during the whole session. The experiment data we collected and analyzed include video recordings of each group, FlexiSketch log files listing user actions with timestamps, and participants feedback from the semi-structured interviews and survey.

3.2 Analysis

Each video was analyzed in two iterations by the article's first author. In the first iteration, the editing behavior of each study participant was coded with a binary function (1 during the time when the participant is touching the tablet, 0 for the rest of the time). In order to filter out fine-scale structures while keeping the important behavioral patterns, we applied smoothing to the results by using discrete time steps of two seconds. The same author coded the conversation between the participants in a second iteration. Starting with the experience from the first iteration, he created a coding scheme. Then, he refined the scheme by analyzing two of the video recordings. He discussed the coding scheme with the other authors and his research colleagues before finalizing it. The final scheme consists of four categories, and each utterance from the workshop participants was put in one of the categories: the *modeling* category contains utterances about the modeling task and the domain model (e.g., "*We have a further actor, professor, who can also upload documents*"). The *semantics* category includes statements and questions about types and the modeling language (e.g., "*What does this element mean?*", "*I'm going to draw and define an actor symbol*"). The *tool* category contains utterances related to tool features and usability (e.g., "*Can symbols be rotated?*", "*You need to hold down the finger to drag and drop*"). The *other* category contains chatter unrelated to the task and tool, e.g., when someone mentions the weather or comments on the drawing skills of someone else.

The first author coded the utterances in all videos. Also, dual coding was performed on one of the videos by involving the second author as an additional coder, and its results were discussed. We synchronized the time codes by allowing deviations of ± two seconds. Then, we measured the inter-rater agreement, ignoring brief utterances that were coded by one author but ignored by the other one (utterances such as "*mhm*", "*ok*", and "*oh*?"). Calculating Cohen's kappa resulted in a value of 0.79, which implies that the agreement level is between "substantial" and "almost perfect", and that the first author's coding possesses a reasonable validity.

After we finished the classification, we looked closer at the *semantics* category and investigated how these utterances relate to the type-defining activities of the participants. We were interested in analyzing how participants communicated their type definitions (e.g., do they discuss or just notify each other about the types they create? Do they talk to their team members before or after creating a type?).

On some tablets, FlexiSketch did not generate log files due to a software bug. The log files from the remaining tablets helped us recognize tool interactions in situations where these interactions were difficult to recognize when analyzing the videos.

We further analyzed the results from the semi-structured interviews and the survey, and we grouped statements to find accumulations and interesting patterns. We also looked for connections between participants' behavior during the experiment and their interview answers.

3.3 Results

Figure 5 shows extracts of the diagrams created by practitioners, together with the defined types.[8] Table 1 reveals how many elements and types are contained in the diagrams.

R1.1: *Phases of simultaneous editing happened in all groups*. In all six groups, phases of silent, simultaneous editing and phases of discussions with or without editing were interleaving. Figures 6 and 7 show when each group member was editing and/or talking. In the practitioner groups, all group members were simultaneously editing during 8.2 to 13.5% of the time, depending on the group. In the student groups, the values were higher with 20.1 and 23.3% for SG2 and SG3, and 51.5% for SG1 (the group of two). The different amount of simultaneous editing between practitioner and student groups correlates with the different amount of communication (see R1.2). We found two special cases regarding the editing behavior: practitioner P4 in PG2 made

[7] Choosing a GUI enabled us to also evaluate the fitness of our tool for diagram types that consist of containment relationships rather than nodes and edges, while it still allowed us to observe how students collaboratively define a modeling notation.

[8] The full diagrams can be found in high resolution at https://files.ifi.uzh.ch/rerg/flexisketch/TeamResults.pdf.

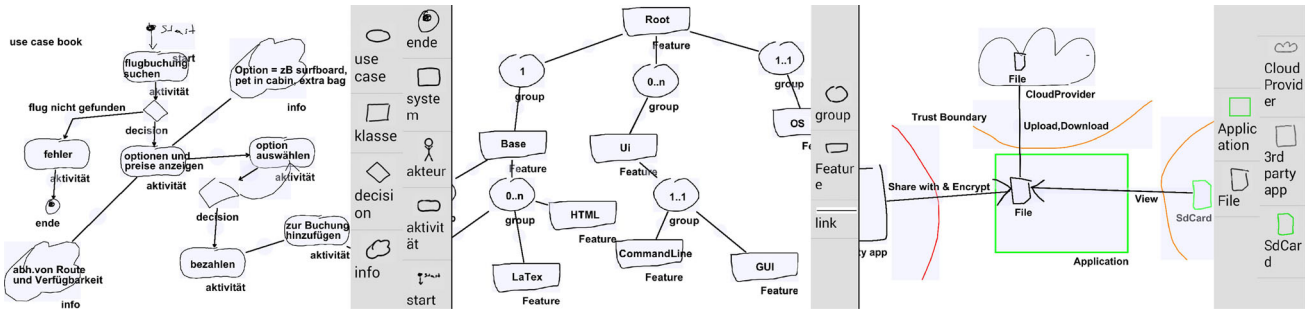

Fig. 5 Extracts from the results of practitioner groups (left: PG1, center: PG2, right: PG3). The gray bars show the defined types

Table 1 Amount of symbols, links, and defined types contained in each diagram from student groups (SGx) and practitioner groups (PGx)

		# Symbols	# Links	# Types
SG1	UC	12	7	5
	GUI	13	0	4
SG2	UC	10	6	3
	GUI	3	0	1
SG3	UC	7	5	4
	GUI	8	0	5
PG1		20	16	9
PG2		18	15	3
PG3		9	3	5

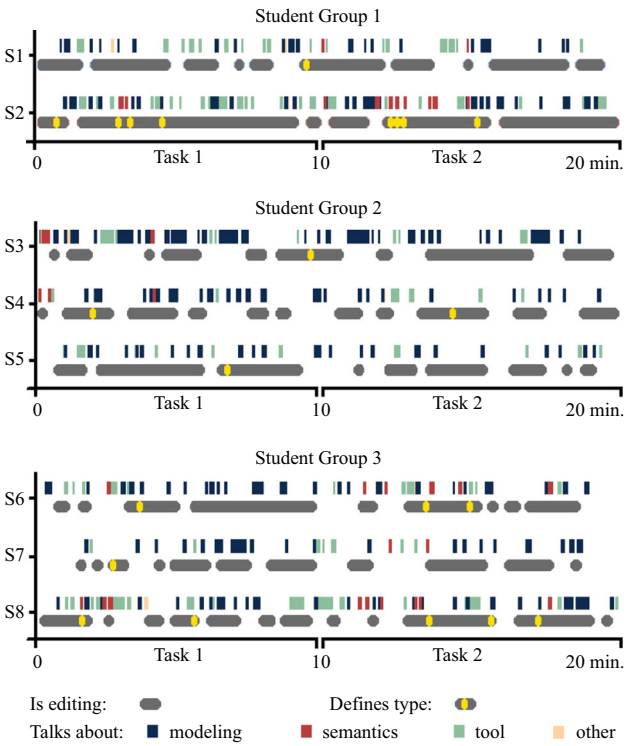

Fig. 6 Phases of editing and discussions in student groups

few edits, instead she contributed to the work by asking many explorative questions and proposed alternatives, e.g., "*Should we have different feature types or just one type called feature?*". Furthermore, we identified student S3 as a leader in SG2. He talked the most and came up with many modeling ideas, while the other members focused on sketching activities.

R1.2: *Practitioners communicated more than students*. We counted during how many of the discrete two-second time steps communication took place, and found that practitioner groups were talking for 12 min on average, while student groups were talking for 7.7 min on average. During the video analysis, we found that practitioners did communicate well: they frequently discussed about what to do, and informed each other about their current or next steps. Practitioners from all groups stated in the interview that they did not experience communication or coordination issues. No one disagreed. In contrast, students from SG1 and SG3 stated that they perceived a lack of coordination, because their attention was drawn to the interaction with the tool. They believed that this reduced the amount of discussions they had, e.g., S2 said: "*Especially at the beginning we did not talk, each of us was concentrating on his own tablet*", and S3: "*Each of us drew something. We only discussed after noticing that two of us had sketched the same thing and we needed to agree about what to keep and what to delete*". Our video analysis confirmed these coordination issues. In this regard, P1 recognized an important difference between a pure sketching environment and our tool: "*When sketching collaboratively with a tool, you can just start to draw. But here, when defining types, you need to be more careful and coordinate*". Sketching can always be done locally, while type definitions affect the whole workspace. (They change the type library and are valid for the whole sketch canvas).

No student group started the task with a brainstorming or extended discussion. Instead, communication happened rather "incremental": multiple times during the session, students discussed what they are going to draw next, and who

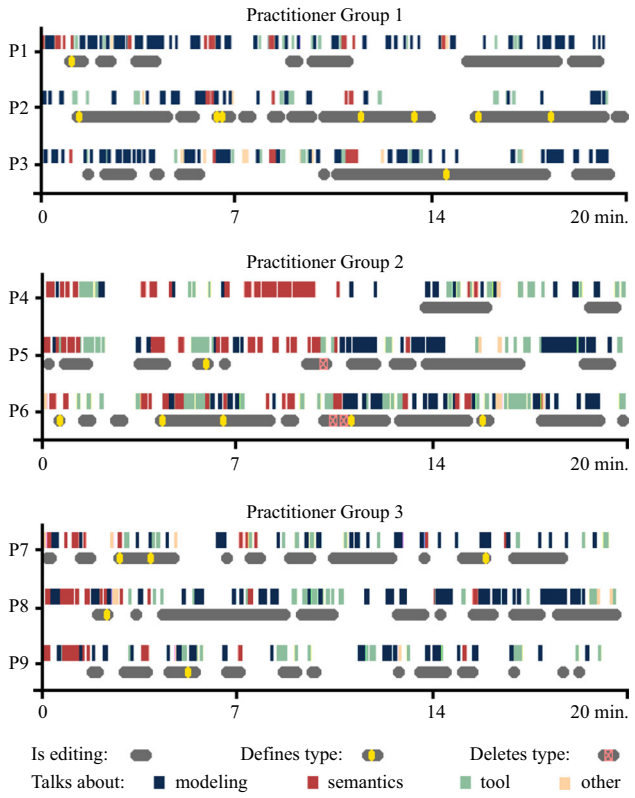

Fig. 7 Phases of editing and discussions in practitioner groups

draws what part. These moments were followed by phases of silent editing.

In contrast to the students, the practitioners almost always informed the other group members about their next steps before drawing anything (e.g., P1: "*I'm going to draw a system boundary, okay?*"). One exception happened in PG1, where communication started to decrease during the session. Toward the end, the group members were simultaneously sketching three different types of diagrams next to each other. To ensure consistency between diagrams, they discussed key elements which appeared in all diagrams, such as specific stakeholders and use cases.

All student groups tried to fit their diagrams on a single tablet screen (such that they always saw all parts without scrolling and zooming). S5 from SG2 stated: "*We wanted to make sure that we always see the changes made by each other, and that no change happens outside of a tablet's current view*". In contrast, diagrams from the practitioner groups clearly extended the size of a tablet screen (except the diagram from PG3 which needs a small amount of zooming to make it fit).

The different groups did use the big screen with the overview to significantly different degrees. While PG1 and PG3 barely looked at it (P1: "*We used it once or twice*"), the video analysis revealed that group members of PG2 used it many times to discuss the design and further steps with the help of a shared view. Feedback from PG2 confirmed this. Likewise, five of the seven students who filled out the survey were very positive or positive about the big screen with the overview (measured on a Likert scale).

Many participants peeked onto each other's tablet from time to time. (The same behavior was found in a study from Loksa et al. [35]). This is true for all students as well as for P1, P2, and all practitioners from PG3. P1: "*It helps to coordinate, to see what the other person is doing*". Only practitioners from PG2 did not reveal this behavior. They were sitting a bit further apart from each other compared to the other groups, which made it hard to see the screens of each other's tablets. Instead, the members of PG2 looked at the big screen more frequently than other groups to discuss the sketch.

R1.3: *Notations were defined by multiple participants*. In all simulated workshops, type definitions were created by multiple group members. Out of all participants, P4 was the only person who did not define any type. Students defined a total of 9 (SG1), 4 (SG2), and 9 (SG3) types, practitioners 9 (PG1), 3 (PG2), and 5 (PG3) types. The yellow dots in the editing bars in Figs. 6 and 7 indicate type definitions.

The video analysis showed that there was no discussion related to the graphical representation of types in all student groups as well as PG1; with one exception in SG3, where S6 communicated that he is about to assign the type *use case* to one of his drawn elements. S8 interrupted him and asked whether they should instead use a nice geometrical shape for the representation, whereupon S6 agreed. Apart from this, only practitioner groups PG2 and PG3 briefly discussed what the individual types should look like (see R1.6 for more discussion details).

R1.4: *Notations were defined incrementally and continuously during the whole sessions*. Most often, groups defined new types as soon as they introduced new elements in the diagram. In all groups, types were not only defined at the beginning, but the notation grew incrementally during the whole task (e.g., Fig. 7 shows that group PG1 defined five out of the nine types during the second half of the session). Practitioner groups PG2 and PG3 discussed many semantics concerns in the early phase of the modeling task and then continued with incremental discussions and ad hoc notation definitions at various points in time. Other groups did not have semantics discussions at the start, but showed the same behavior of discussing the language incrementally.

R1.5: *Participants based their notations on familiar concepts and symbols*. All practitioner groups ended up with nonstandard modeling languages which are more (PG1) or less (PG6) based on existing standards (such as UML or feature tree models). Participants from PG1 and PG2 stated in the interview that their groups chose and agreed on language concepts that were familiar to all group members, and then started to adapt and augment them as needed. For example,

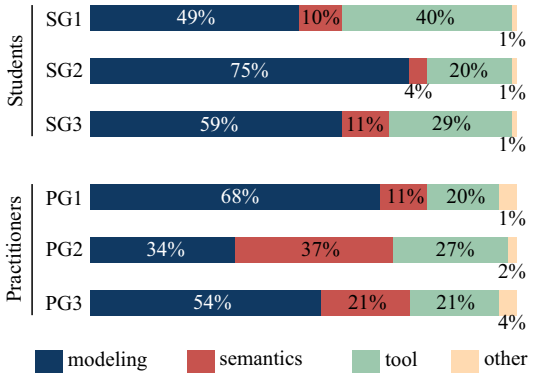

Fig. 8 Talk category distribution in student and practitioner groups

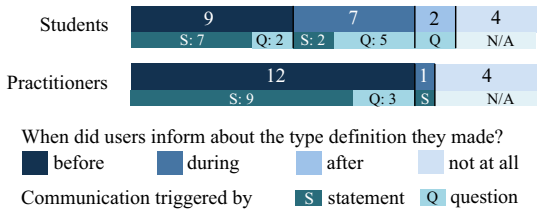

Fig. 9 When participants talked about new types—before, during, or after defining them

PG1 started with a basic UML activity diagram notation and then augmented it with an additional element type during the session, an *info* type that is represented with a cloud-like shape. As another example, PG2 started with the idea of a feature tree model. They did not augment it, but ended up with a simplified version of the model type. Finally, PG3 came up with a DSML. Thus, we experienced three cases: (i) augmenting a standard notation, (ii) simplifying it, and (iii) coming up with a custom notation. The common result is that none of the practitioner groups did stick to an unaltered, standard notation.

R1.6: *Discussions about semantics depended on the chosen language constructs*. Basing custom languages on standards also simplified the task of achieving a mutual understanding about the meaning of individual elements. Figure 8 reveals that PG1, using a notation which was very similar to UML, devoted less of its discussions to semantics than PG2 and PG3. Four types in PG1 were defined by P2 without any discussion.

PG2 discussed a lot about semantics and how they can map their concerns to model elements. (They talked more about semantics than the concrete model). It was the only group that deleted some element types in the middle of the session and replaced them by a new one (see Fig. 7). The recording of the discussion revealed that they found the old types to be at a wrong level of abstraction: at first, they did not define the type *group*, but multiple different subtypes of it. Later in the session, they decided that they do not need this level of detail, deleted the subtypes, and replaced them by the single *group* type.

Student groups talked little about semantics. P2 mentioned: "*There was no need to discuss because we were all familiar with the use case diagram notation needed for the first task*". The resulting sketches show that all groups used the proper notation for use case diagrams. In contrast, the notation for the GUI in task two was not predetermined. While creating the GUI for task two, SG1 and SG3 talked more about semantics (see Fig. 6), but SG2 almost completely neglected semantics. Indeed, they only defined one type during task two (Table 1).

Figure 9 reveals how participants communicated about type definitions.

For nine out of 22 types, students discussed their type definition in advance (before they created it in the tool). In nine other cases, they only informed their group members by quickly mentioning the symbol type, either while inputting the types or only afterward. (For example, S6 and S8 mentioned at the moment when they hit the *ok* button of the type definition dialog: "*Radio button*" and "*Text field*". respectively). Seven of these types were only discussed after another group member asked the creator of a type what he/she is doing or what the newly appeared type means.

In contrast, practitioners discussed many type definition in advance (twelve types). Three of these discussion started with a question (e.g., P8: "*Should we define a type named file?*"), the others with a statement. No one felt the need to ask a question about an already defined type, except in PG2, where types got re-discussed after they were in use for some time.

Both student and practitioner groups did not discuss four type definitions at all. For practitioner groups, these four cases happened in PG1. After agreeing on the UML activity diagram to start with, they did not feel the need to discuss some types individually. P1: "*Borrowing most of the elements from UML allowed us to get a shared understanding of the symbols' meaning with little effort*".

R1.7: *All groups created consistent notations*. The survey revealed that the drag and drop mechanism for reusing types was liked a lot by the study participants. The video analysis and the resulting diagrams made clear that students and practitioners used the mechanism whenever possible. (With one exception in PG1, where Fig. 5 shows two slightly different hand-drawn symbols for the type *decision*). Making heavy use of the drag and drop mechanism resulted in all participants using the same notation within a group and therefore led to consistent notations. Also, the video analysis and log files provided no evidence that temporary inconsistencies happened during the workshops. In the end, almost all diagrams from students and practitioners showed a 1:1 mapping between element graphics and types. Apart from

two exceptions, there were no inconsistencies such as either having two different shapes with the same type assigned, or two identical shapes with different types assigned. One exception happened in PG1, where two symbols with different types (*system* and *class*) cannot be distinguished by the sketch recognizer, because they look the same. P2, responsible for creating the type *class*, told us in the interview: "*I wanted to sketch a better class symbol, but refrained from it because the session was about to end*". Other than that, all symbols can be distinguished by form and/or color. The second exception happened because of the tool's inability to rotate symbols. PG3 needed multiple *trust boundary* symbols with different orientations. P8 stated that "*the possibility to reuse defined types is a big motivation for defining them*". Not being able to rotate the symbol, P8 had no motivation for creating a type definition: "*Can I rotate a symbol? ... No? ... In that case, I do not need to assign a type to this particular symbol*". However, P5 and P9 stated another advantage of defining types: assigning types to symbols can serve as a kind of documentation and contribute toward the comprehensibility of a sketch. P5: "*Due to the type definitions, I think I will have less effort in understanding a sketch when I look at it again after several weeks or months*".

On a Likert scale, all students were very positive about the drag and drop mechanism. S2 said: "*As soon as you start to make bigger sketches, dragging elements [from the type library] onto the canvas is faster than drawing them by hand each time*".

Regarding completeness, PG3 did not define the *trust boundary* symbol, and SG2 did neglect definitions for the GUI elements. Apart from these cases, the type definitions for symbols were complete, i.e., every symbol in the sketches had a type assigned to it. In contrast, PG2 was the only group that defined a link type. Other groups neglected link types. We discuss possible explanations for this in the discussion Sect. 5.1.

4 Study 2: to what extent can modelers define lightweight modeling languages correctly and completely?

With RQ2, we want to investigate whether potential users of our approach manage to define modeling languages correctly and completely on their own. In a study, we conducted a quantitative experiment with 107 first-year undergraduate students from two universities (67 from the University of Zurich and 40 from the University of Applied Sciences and Arts Northwestern Switzerland FHNW) and complemented it with a qualitative experiment including eleven practitioners from different companies. The main goal of the student experiment was to find out whether students provide correct and complete lightweight metamodel definitions when using the metamodeling mechanisms of our approach. If they manage to do this, it would mean that we can create this kind of lightweight metamodels without the help of metamodeling experts. The main goal of the practitioner experiment was to see how useful practitioners find our tool, and how well they manage to apply our approach. The practitioners have various degrees of modeling and metamodeling experience. Table 2 provides an overview of the practitioner demographics.

Five practitioners work as software engineers and mainly conduct software design, architecture, and development. The other six are specialized in requirements engineering. The practitioners had a broad range of experiences in SE and RE, including practitioners who were working in software and consultant companies for about 2 years after finishing their studies, as well as a practitioner aged 60, having more than 30 years of experience. Five practitioners had experience with metamodeling, and four of them had already used metamodeling explicitly in their jobs. The other six had no metamodeling knowledge. The students had basic modeling and no metamodeling knowledge and thus can be seen as novice modelers. They can be considered to be future software and requirements engineering practitioners, and therefore our future target audience. Furthermore, their level of modeling knowledge might be comparable to those of many business stakeholders.

4.1 Method

To be able to assess the quality of the students' solutions in terms of correctness and completeness, we specified the modeling task and the language that students had to use. This enabled us to compare their solutions to a ground truth (which represents an expert solution).

In contrast, in order not to limit our experiments to specific modeling languages and tasks, practitioners were free to choose their own languages and could model aspects of projects they were currently working on. Although this decision does not allow us to compare student results directly with practitioner results, it provided a much more realistic evaluation scenario (that was not possible with students due to their lack of experience).

4.1.1 Student experiment

Setup. From the 107 students, 64 students solved the experiment tasks with the mobile version of our tool on an Android tablet, while we randomly assigned 43 of them to a control group where they solved the same tasks with paper and pencil. Since we do not only want to evaluate our tool but also our approach, this would allow us to identify a possible bias introduced by the tool itself.

The experiment lasted 60 min and was structured into three parts: briefing, evaluation, and debriefing. The briefing

Table 2 Practitioner demographics: their work field (SE/RE), the years of work experience, experience with touch devices (high (H)/medium (M)/low (L)), and metamodeling knowledge (yes/no)

Practitioner	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
Work field	SE	RE	RE	RE	SE	RE	RE	SE	SE	SE	RE
Years of work experience	10	15	> 30	8	2	2	11	23	9	4	3
Experience with touch devices	H	M	L	M	H	M	M	H	M	L	M
Metamodeling knowledge	Y	Y	Y	N	N	N	Y	N	Y	N	N

started with a short oral introduction to inform the students about their modeling task. In addition, the students got handouts which provided a textual description of the example they were asked to model. (The material is available online).[9]

For the students who used FlexiSketch, the handouts also included a short hands-on tutorial to get to know the tool's functionality. No introduction to metamodeling was included. The handouts only stated that the tool must be able to interpret the sketched diagrams, and that therefore all model constructs must be defined. It was made explicit that there is a passive wizard that can help to define all model elements and cardinalities.

The students who did the experiment on paper had to solve the same examples without tool support. The handouts stated that they should define all modeling constructs so that a person without knowledge about these diagram types can understand them. Their task was divided into three parts. Each part was presented on a new page: first, students had to sketch the diagram on paper. Second, they had to build a table with all sketched symbols and links in one column, and proper type names in another column. Finally, they had to document cardinalities for each defined link type.

During the actual evaluation, students modeled UML diagrams based on the given textual scenarios and provided metamodel information by defining symbol types, link types, and cardinalities. The students from the University of Applied Sciences and Arts Northwestern Switzerland FHNW had to model a use case diagram about a simple railway scenario including two actors, four use cases, and three types of relationships: association, includes, and extends. The students from the University of Zurich had to model a class diagram based on a related railway scenario. This included five classes, two note objects as introduced in the handouts, and three types of relationships: association, inheritance, and describes. The latter was used to connect a note object with a class.

In total we received 107 diagrams including metamodel information, as summarized in Table 3.

At the end, all students were asked to fill out an online questionnaire for debriefing purposes.[10] The questionnaire included questions on their personal skills and their experience during the evaluation task.

Table 3 Amount of data points received

	With FlexiSketch	With paper
Class diagram	34	33
Use case diagram	30	10

Data analysis. In a first step we assessed the quality of a student's model, comparing it to our ground truth[11] in terms of completeness and correctness. (We created the ground truth for the models and the metamodels based on our textual scenarios). Since our lightweight metamodeling approach consists of metamodeling by example, a wrong model can affect the resulting metamodel. We wanted to take these aftereffects into account, in order to receive values for the metamodel quality that are independent of the model quality. We differentiated between syntax and semantics. A semantically incomplete or incorrect model can still lead to the same metamodel if the syntax is complete and correct. But if there is a problem with the syntax, the metamodel will look different. In cases where the student's syntax differed from our ground truth, we evaluated the student's metamodel against his/her model (and not the model from the ground truth) to rule out consequential errors. With this approach, we also respect that there might be other correct model solutions for the given task, apart from our ground truth.

We then measured the completeness and correctness of a student's metamodel by comparing it to our ground truth. We independently looked at type definitions and cardinality definitions for measuring completeness. To measure type definition completeness, we counted how many of the symbol types and link types found in a student solution were defined by that student (#DefinedTypes/#TypesInSketch). For the completeness of cardinality definitions, we checked how many of the link types that were defined by a student also had cardinalities assigned. Due to how our tool works, each link type has either none or four cardinalities assigned (minimum and maximum values for both ends of the link; if the user does not define all four cardinalities, the tool infers the remaining ones). We then counted how many of the stu-

[9] https://files.ifi.uzh.ch/rerg/flexisketch/Handouts.pdf.

[10] https://files.ifi.uzh.ch/rerg/flexisketch/Questionnaire.pdf.

[11] https://files.ifi.uzh.ch/rerg/flexisketch/GroundTruth.pdf.

dent's definitions were correct. For symbol and link types, we considered them to be correctly defined if the wording either matches the official UML definitions, or if synonyms were used that have the same meaning (as judged by two of the article's authors). For cardinalities to be correct, the values had to exactly match our ground truth. For the correctness metric, we ignored type definitions for additional model elements that were introduced by a student if neither the sketched model element nor the accompanying type definition is part of our ground truth for the model and the metamodel. (15 out of 107 students each introduced one such model element and respective type). For example, one student defined the additional type *extension point* for the use case diagram and used one instance of it in the model. Assessing the correctness of such additionally introduced types would be difficult in many cases (e.g., when the types do not match those of the official UML standard), because the intention of the modeling person would have to be taken into account, which can lead to highly subjective judgements.

Apart from the models and metamodels which were part of the student solutions, we also investigated the log files that FlexiSketch produced during the experiment. The logs show a list of user actions with timestamps. This allowed us to see, e.g., if the app crashed, how many times a student used a functionality, and if the same symbol was redefined several times. This kind of information cannot be deduced from the final student solutions. In combination with the student answers to the online questionnaire, this allowed us to reason to a certain extent whether some errors in the student's solutions are due to a lack of the student's knowledge or usability problems of FlexiSketch.

4.1.2 Practitioner experiment

Setup. The setup was similar to the student experiment in order to allow for a comparison of the results. However, practitioners were asked to sketch a model about a real-world task or problem they currently experience in one of their companies' projects and to use any node-and-edge notations—as they see fit—for the model. This could include established notations such as UML, but we emphasized the tool's openness toward custom notations in the introduction.

We first introduced FlexiSketch in a semi-structured way, following the tutorial from the student experiment. We then asked the practitioners to sketch a model that is typical for their model sketching activities in the context of their job. They should furthermore define all types and cardinalities.

Because very few students used our tool's wizard in the student experiment (see Sect. 4.2), we explicitly advised practitioners to consult the wizard after finishing their modeling activity. After the modeling task, we conducted a semi-structured interview where we asked the questions from the student experiment questionnaire. Additionally, we asked more specific questions about the usability and utility of the tool in the context of the practitioners' work.

Data analysis. As the practitioners used their own custom modeling notations, creating the ground truth for assessing the correctness of the created metamodels is not possible, or at least very difficult. But this was not the main goal of the practitioner experiment. Instead, we were interested to see whether practitioners think that our approach is useful for the types of modeling notations that they use (having a reality check for our tool), and whether they face similar difficulties compared to students, or if they have no problems in providing metamodel information.

Nevertheless, we also tried to achieve a best approximation to a metamodel ground truth (i.e., the practitioners' true intentions about the meaning of what they had sketched) by discussing the experiment results (e.g., the derived metamodels) with the practitioners. This delivered some pointers for judging metamodel correctness, but inherently contains partly subjective views (which might also change over time). We also asked the practitioners to draw their intended metamodel on paper. In contrast, we counted the amount of defined and undefined symbols to have an objective measure for metamodel completeness.

Most of the presented results from this experiment are based on an analysis of the semi-structured interviews.

4.2 Results

4.2.1 Students–modeling novices

R2.1: *Models were of good quality, although not all were complete*. In general, most students delivered models of good quality (as shown in Table 4). While many models were missing some parts (the average model completeness was 75.9%), most of the drawn elements were correct with an average of 91%.

Model results between paper and FlexiSketch exercises were similar, except for the completeness of the class diagrams (82.1% for paper solutions and 66.1% for FlexiSketch solutions). We performed a two-sided Welch's t-test and found that the difference is statistically significant (p value: 0.002) for the class diagram completeness, while the other differences in model quality are not statistically significant (Table 5). On tablets, fewer students used the additionally introduced *note* symbol and the accompanying link type *note-link* when creating the class diagram. (This is shown in Fig. 10).

Also, some students with tablets depicted association and inheritance relationships with the same visual link style.

For the use case diagram, many students did not sketch all association relations between actors and use cases, which reduced model completeness (for both tablet and paper solutions).

Table 4 Completeness and correctness of student models and metamodels (without cardinality rules)

Diagram type		Model (%)		Metamodel (%)	
		Compl.	Corr.	Compl.	Corr.
Class	Paper	82.1	93.9	77.4	85.8
	FlexiSketch	66.1	95.8	73.8	60.4
Use case	Paper	77.8	89.0	95.6	83.5
	FlexiSketch	77.4	85.3	70.6	95.3

Table 5 Statistical significance of the differences between paper and FlexiSketch results

p values	Model (%)		Metamodel (%)	
	Compl.	Corr.	Compl.	Corr.
Class diagram	0.002	0.38	0.64	0.004
Use case diagram	0.94	0.50	0.00006	0.096

R2.2: *Metamodel quality differed between type and cardinality definitions*. Table 4 shows the average completeness and correctness of the metamodels. These values were calculated by counting symbol and link types, but not cardinalities. Students' metamodeling performance varied significantly between element types and cardinality definitions. Therefore, we look at these results independently (type definition results are covered in R2.3 to R2.5, cardinality definitions in R2.6).

R2.3: *Metamodels were more complete on paper*. Table 4 reveals that metamodel definitions done on paper tended to be more complete. The difference for the class diagrams is not statistically significant ($p = 0.64$), while there is a significant difference for the use case diagrams ($p = 0.00006$, Table 5). Many students using FlexiSketch did omit a definition for the *association* relationship in the use case diagram, which is the main cause for this difference. Furthermore, the *actor* type is lacking in some solutions.

In terms of completeness of the FlexiSketch solutions, we were not only interested as to whether all types were defined and appeared in the type library, but also whether all elements on the sketch canvas had a type assigned. FlexiSketch log files show that most students defined an element type right after sketching a new element for the first time. Afterward, they used the drag and drop mechanism of our tool to create more instances of the same type. Therefore, in most of the FlexiSketch solutions, all elements on the sketch canvas were defined. Students who did not use the drag and drop functionality assigned the type to each instance by hand. There was only one solution where some instances of the same type did not have the type assigned to them (i.e., an incomplete assignment of one particular type to all its instances). For the rest, either all instances of a type were correctly typed, or none of them had a type assigned in the first place.

R2.4: *Metamodel correctness varied between diagram types*. Evaluating the correctness of the different metamodels resulted in diverse values.

Figure 10 presents a more detailed view of the completeness and correctness of students' metamodel definitions. For every element type, it shows from left to right: how many students sketched this type in the diagram (blue), how many students provided a definition for the element type (red), and

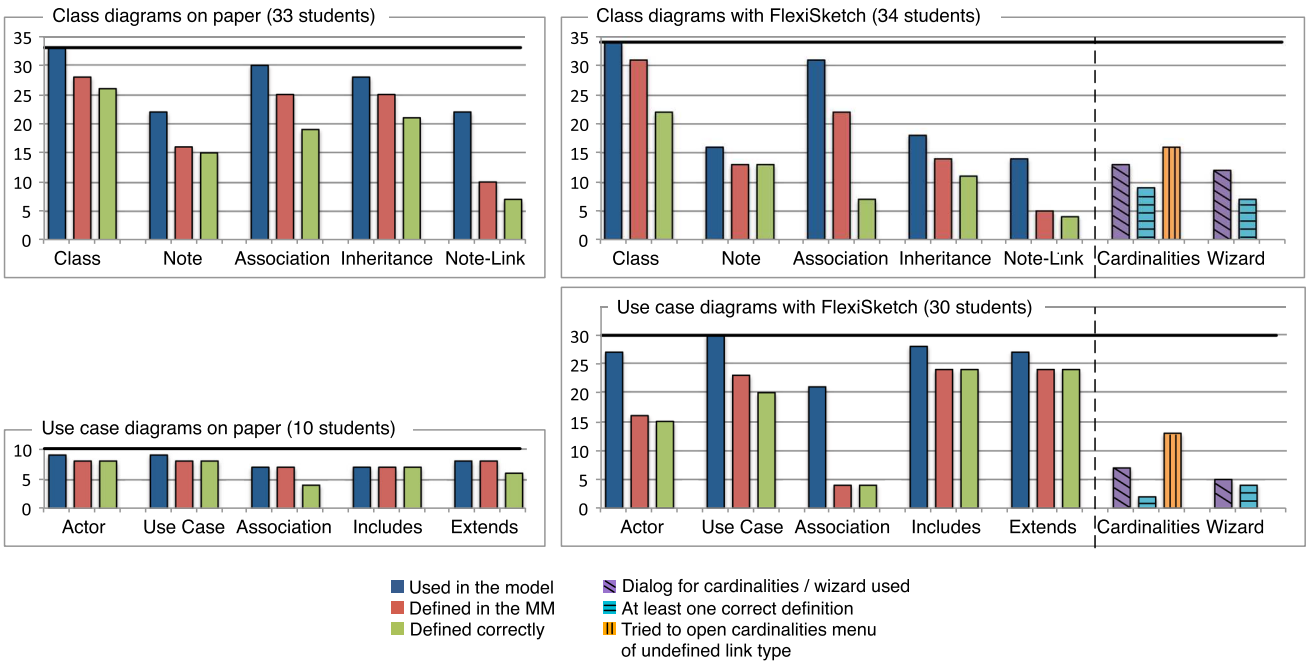

Fig. 10 Amount of metamodel information defined by students

how many of these definitions were correct (green). Complete and correct type definitions would mean that all three bars of a group have the same height. For the students who used FlexiSketch, the figure also shows how many students used the cardinality dialog and the wizard at least once (violet), and how many provided at least one correct definition with the respective dialogs (cyan). Additionally, the amount of students who tried to define cardinalities for an untyped link is shown (orange).

While students with FlexiSketch performed very well in defining use case diagram elements (95.3% correctness on average; the second and third bar of each group in Fig. 10 have similar or same heights), students defining class diagrams with FlexiSketch were less successful (60.4% correctness). Fifteen out of the 22 students who did provide a definition for the *association* relationship did provide a wrong definition, resulting in the biggest difference between red and green bars in Fig. 10. For example, some of these students named the relationship simply *arrow*, or they defined it on the model level as *serves*, which only makes sense for that particular instance of the association relationship. This problem was less present for students who did it on paper (six out of 25 students). There is a clear significance regarding the difference in paper and FlexiSketch metamodeling results for class diagrams ($p = 0.004$). On the other side, FlexiSketch results for use case diagrams are better than paper results, although not statistically significant ($p = 0.096$) unless we lower the level of confidence to 90%.

We also counted how many of the definitions correctly describe element types on the metamodel level (independent of whether it is the correct type name or not), and how many of the definitions were instead done on the model level, describing particular instances of the types (e.g., having types named *serves* and *contains*, instead of having a type definition named *association* for the respective relationship type in the class diagram). Regarding the use case diagram, 33 out of 38 definitions on paper and 89 out of 91 definitions created with FlexiSketch were on the metamodel level. For the class diagram, 99 out of 104 definitions on paper and 68 out of 85 definitions created with FlexiSketch were on the metamodel level. Many of the remaining 17 definitions were describing instances of association relationships rather than the relationship itself. Six students reported in the post-experiment survey that they had problems with assigning different types and/or cardinalities to different instances of the same relationship type. This confirms that they did not think on the metamodel level in these cases.

R2.5: *Students with tablets provided more accurate type names.* As far as the accurate naming of types is concerned, FlexiSketch solutions tended to be superior (p 0.037) to paper solutions: on paper, more students did not provide the exact type names but variations and paraphrases (e.g., on tablets, the stickman type was almost always named "actor", while on paper it was sometimes named as "user" or "system user". Also, instead of using the precise names for associations, they were sometimes paraphrased, e.g., "arrow for extends relationship" instead of "extends"). Taking into account only correct type definitions, tool solutions yielded 125 precise definitions out of 144 (86.8%), while paper solutions yielded 86 precise definitions out of 124 (69.4%).

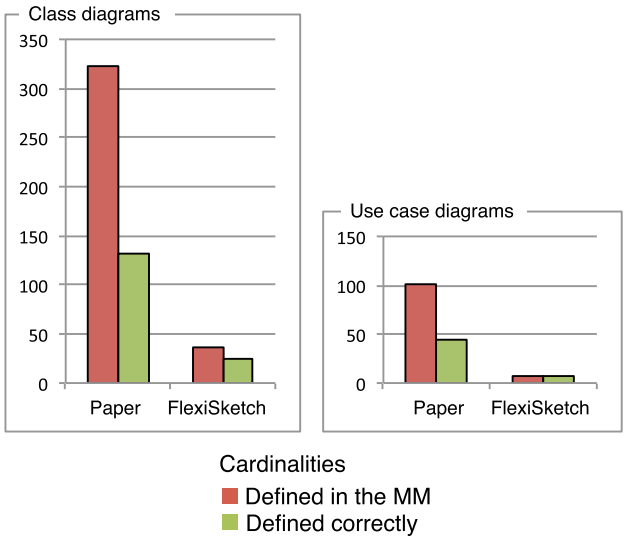

Fig. 11 Amount of cardinalities defined by students

R2.6: *The cardinality and wizard dialogs were rarely used.* Figures 10 and 11 show that the cardinality dialog and the tool wizard were used by relatively few students. Thirteen students modeling the use case diagram and 16 students modeling the class diagram tried to access the cardinality dialog before they defined the respective link type. This is something that the tool does not allow—the link type needs to be defined before cardinalities can be specified. This means that the number of students who used the cardinality dialog would have been significantly higher if this tool limitation would not have existed. Whether these students would have provided correct cardinalities or not remains unknown.

Most students who did the experiment on paper provided cardinalities. For the use case diagram, 44.1% of the cardinalities were specified correctly. Three out of ten students provided cardinalities for link instances instead of link types. For the class diagram, 40.7% of the provided cardinalities were correct. Twelve out of 33 students provided cardinalities for link instances. With FlexiSketch, much fewer students (about eight times less) defined cardinalities. Therefore, the red bars in Fig. 11 are much lower for FlexiSketch solutions compared to paper solutions. However, 75% of the cardinalities provided with FlexiSketch were specified correctly, which is a much better correctness value compared to paper. But this result could be biased by the inability of many students to access the cardinality dialog (i.e., if we assume

Table 6 Comparison of times between students working with paper and FlexiSketch

Times		Average	SD	*p* value
Class diagram	Paper	0:22:11	0:08:30	0.52
	FlexiSketch	0:20:43	0:07:37	
Use case diag.	Paper	0:20:06	0:06:18	0.86
	FlexiSketch	0:19:39	0:07:32	

that only the best students managed to access the cardinality dialog, then it is not surprising that we obtained a better correctness value).

In total, only 17 out of 64 students consulted the wizard during the experiment. Some students reported in the post-experiment survey that the wizard is not needed, because everything can be done without it. Others said that they simply forgot about the wizard's existence.

R2.7: *No time difference between tool and paper emerged.* Before and after their main task, students had to write down the current time. Analyzing the times, we found no significant differences between the averages of paper and FlexiSketch solutions, with *p* values of 0.52 (class diagram) and 0.86 (use case diagram), as shown in Table 6.

4.2.2 Practitioners—expert Modelers

The practitioners drew various models, resembling use case, context, component, and activity diagrams, but overall being simpler in their notation compared to UML standards. Three sketches combined process diagrams with static views such as use case diagrams. One model only consisted of boxes and arrows (as judged by its creator). We were expecting this kind of simple notations. This result is inline with a known design behavior, stating that "*designers draw what they need, and no more*" [39]. Moreover, a notation that is too complex can stifle creativity [4].

R2.8: *Practitioners outperformed students considerably in terms of metamodel completeness and correctness.* The practitioners have more (meta)modeling experience than the students and showed a faster understanding of the respective tool features. Metamodels from practitioners turned out to be superior compared to students' metamodels in both completeness and correctness. Regarding correctness, we only identified mistakes in three metamodels: one practitioner failed to define element types on the metamodel level. Another practitioner used the same link appearance to depict three different relations; therefore, the type definitions were not correct. Furthermore, one practitioner did not care about the semantics of a link and simply assigned the type *arrow*.

Five practitioners handled type definitions the same way as the students in our previous experiment: they defined elements when they sketched them for the first time. Afterward, they used the drag and drop mechanism to create more elements of the same type. Therefore, all elements of the same type were already automatically defined.

We told all practitioners to use the wizard at the end of the modeling task. But before we said this, we stored the solutions from the practitioners, such that we could compare these results with the student results (and not the results that include the improvements with the help from the wizard). The completeness of metamodels before practitioners used the wizard included the full range from 0% to 100%, as shown in Table 7. After consulting the wizard, only one metamodel was not complete. All symbols in the sketches were defined, except in one case, where a practitioner drew about 20 symbols conveying the same meaning by hand, and then did not add types to all of them.

R2.9: *Different opinions about the usefulness of metamodel features.* Almost all practitioners tried to define cardinalities. Seven out of eleven practitioners succeeded in providing 100% correct cardinalities. Four practitioners stated that the ability to define cardinalities is relevant, while the others stated they do not need cardinality definitions for how they would use our tool in practice. They said that it is more important to assign meanings to the sketches in the form of element types in order to discuss the sketches with coworkers and business stakeholders and to reach a common understanding, rather than providing all necessary data for completely formalizing the sketches.

Six out of eleven practitioners said that the wizard is useful and gave tips for improvements, while five did not feel the need for a wizard. On the other hand, all but two did in fact use the wizard to define at least one additional element type. One practitioner preferred to use the wizard for all metamodel definitions and therefore did not use the context menu icons of sketched elements.

Practitioners said that the wizard can be useful to complete the metamodel, to delete superfluous elements from the sketching canvas, and to scrutinize the sketched model.

5 Discussion

In this section, we discuss the results and answers to the two research questions, derive a set of guidelines for flexible modeling tools based on the findings, and mention threats to validity.

5.1 RQ1: What patterns emerge when modelers collaboratively define lightweight modeling languages?

The patterns found in the first study (Sect. 3) are summarized in Table 8. Results show that all participants except one took an active part in defining element types (R1.3). Partic-

Table 7 Completeness (%) of practitioners' (p) metamodels before (v1) and after (v2) consulting the wizard

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
v1	80	100	66.7	75	100	25	33.3	60	0	57	75
v2	100	100	100	100	100	100	80	100	100	100	100

Table 8 Summary of the patterns found in study 1

R1.1:	Phases of simultaneous editing happened in all groups
R1.2:	Practitioners communicated more than students
R1.3:	Notations were defined by multiple participants
R1.4:	Notations were defined incrementally and continuously during the whole sessions
R1.5:	Participants based their notations on familiar concepts and symbols
R1.6:	Discussions about semantics depended on the chosen language constructs
R1.7:	All groups created consistent notations

ipants oriented themselves by standard modeling languages but introduced additional types as needed (R1.5). Dekel and Herbsleb found the same, deliberate deviation from standard languages in their study [9]. The modeling languages were further created incrementally during the whole workshop sessions and discussions about semantics happened during the whole sessions (R1.4). This behavior is exactly what our approach is meant to support and foster. Furthermore, we experienced that phases of simultaneous, silent editing did interleave with phases of discussion (R1.1 and R1.2). Other studies have found the same design behavior of group members switching between synchronous and asynchronous work [39].

The results also show that user awareness is an important matter in a multi-screen setup even for same-place collaboration (R1.2). This is something we neglected thus far, except for the visualization of the locking mechanism. Indeed, students sometimes experienced communication and coordination issues because they were mainly concentrating on the tool. In other words, students did not always succeed in managing both the cognitive and the social space at the same time [33]. Shih et al. [51] confirm our results and state that users do not automatically "develop a sense of tolerance for lack of social awareness" in collocated sessions. However, studies suggest that people can learn how to cope with a multi-space setting [35]. Indeed, practitioners did not report coordination problems, and our video analysis shows that they were able to focus on both the communication and their individual screens. However, they mentioned that splitting the focus does require additional cognitive effort.

The frequent use of the drag and drop mechanism had some positive effects on the results in terms of notation consistency (R1.7). Although communication was sometimes lacking in student groups, they seemed to be eager to use the drag and drop mechanism and therefore regularly checked the type library for new types instead of hastily defining new types which could have let to potential duplicates. We also assume that a frequent use of the drag and drop mechanism fosters diagrams that are defined more completely, since each reused element already has a type assigned, compared to the scenario where a user draws a lot by hand. (However, a future study has to confirm or reject this hypothesis). On the other side, the drag and drop mechanism also had a side effect: participants committed to element types early. Related studies [9,43] show that the meanings of symbols are re-discussed and changed during design sketching on physical media. This did not happen often in our study, neither in the student groups with UML, nor in the practitioner groups where participants could freely choose their modeling language. Maybe the drag and drop mechanism tempted the participants to a premature commitment regarding the meanings of drawn elements [43]. (However, the relatively short duration of our simulated workshops could be another explanation why participants did not re-discuss element types).

In contrast to defining symbols, only one group (PG2) defined a link type. Possible reasons could be that link types cannot be dragged and dropped, and that FlexiSketch implicitly keeps a 1:1 mapping by regarding all links with the same appearance as being of the same (undefined) type. Overall, our tool is an example of how a sketching tool can help to have consistent and unambiguous sketches at the end of a session if the users want this.

Compared to the quantitative study where students worked individually (Sect. 4), type definitions from groups were much more complete. Whether this is because participants in the simulated workshops were more aware of their task, or because they were working in groups, remains open and cannot be answered with the data we have.

5.2 RQ2: To what extent can modelers define lightweight modeling languages correctly and completely?

The results of our second study (Sect. 4) are summarized in Table 9. Practitioners had no problems using our metamodeling mechanisms. In contrast, students were able to define types correctly, but sometimes failed to distinguish between the model and the metamodel level when adding cardinality rules (R2.2).

Table 9 Summary of the results from study 2

R2.1:	Models were of good quality, although not all were complete
R2.2:	Metamodel quality differed between type and cardinality definitions
R2.3:	Metamodels were more complete on paper
R2.4:	Metamodel correctness varied between diagram types
R2.5:	Students with tablets provided more accurate type names
R2.6:	The cardinality and wizard dialogs were rarely used
R2.7:	No time differences between tool and paper emerged
R2.8:	Practitioners outperformed students considerably in terms of metamodel completeness and correctness
R2.9:	Different opinions about the usefulness of metamodel features

5.2.1 Students—Novice Modelers

Model quality. In general, the correctness of the drawn models was very high (R2.1). This shows that the students knew the notations and how to model. The lower values for use case diagrams are mainly an effect of students making the arrows of *includes* and *extends* relationships on the wrong end of the links. Analyzing the resulting sketches, we can conclude that most of the omissions and errors done in the models have little effect on the ground truth for students' metamodels.

Metamodel quality. Regarding the metamodel quality, we have to distinguish between element types and cardinality rules. The students performed relatively well in defining element types. The difference in completeness between paper and the tool (R2.3) could be explained by the handouts for the students. The paper version presented each task on a new page (modeling, type definitions, and cardinality definitions) with empty space for solutions, which might have caused these students to care more about providing complete definitions. In contrast, the tablet version of the handout did not have this detailed structure, but mentioned that the wizard can be used to ensure completeness of the language definition. The wizard would then sequentially go through all missing definitions, providing the students a similar help compared to the paper handouts which listed the tasks on separate pages. However, many students did not use the wizard (R2.6). Possible reasons could be that they did not care much, or that they were too self-confident and did not think that they would need the wizard. By not using the wizard, students with tablets also missed the opportunity to be reminded of cardinality rules definitions. Therefore, we suspect that our handout text was flawed by not urging the use of the wizard enough, which led to most of the differences between paper and tool solutions.

Figure 10 shows that students with our tool in particular had problems with defining the *association* relationships in both diagram types. A possible reason for the few defined *association* relationships in use case diagrams is that textbook examples depict *association* links between actors and use cases as simple lines without text, while *includes* and *extends* relationships are usually annotated with the respective names. Thus, students might not have been aware of the correct type name, or that they should assign a type at all to the *association* relationship. A similar argument could explain the difference between the amount of correctly defined *actor* and *use case* symbols: the name "use case" is more present as it is also the name of the diagram. The class diagram contained an additional pitfall compared to the use case diagrams, which caused a bit of confusion among the students with tablets (R2.4): they had to use the same link type (the *association* relationship) multiple times in the diagram and add a different text label to each instance. Many students tried to add the text by defining the link type, rather than using the separate text box functionality. Therefore, students modeling the class diagram with FlexiSketch had more definitions on the model level compared to students who did it with paper. This could be improved in the future by improving the tool's usability and making the difference between the text box and the typing feature more clear.

Although some of the results were worse when compared to paper, we found two advantages of our tool over paper. First, the tool seemed to help keeping definitions short and precise (R2.5), so that more of them could be mapped 1:1 to the official UML names. In the future, this would facilitate the automation of an export/conversion of the sketch to other tools and languages. Maybe the students with tablets were aware of the fact that they are "teaching" the tool a language (as it was stated in the handouts), and therefore were motivated to give precise names. The precise wordings are also good for standardization and for reaching consensus in combination with an additional glossary that defines the terms. Second, the drag and drop functionality motivated students to define types and helped them to provide definitions on the metamodel level (although at the caveat of stifling creativity). When reusing types from the type library via drag and drop, it became clear to the students that reusing an element of type *class* is much more helpful for creating a class diagram than reusing an element of type, e.g., *train* or *train station*. Thus, the type library enabled the students to verify that their type definitions are at the right level of abstraction.

Such a verification mechanism does not exist when defining cardinality rules. Defining such rules had no visible effect on the modeling task, as the tool did not enforce the adherence to cardinality rules. (However, our tool is designed with the idea that users can optionally enable a highlighting system that shows any violation of the cardinality rules). In addition, the tool did not allow students to assign cardinalities before assigning types to the link and the connected symbols

(since cardinalities hold for specific relationship types).[12] However, many students tried to open the cardinality dialog too early and gave up in the process. These two problems might explain the difference in the amount of defined cardinalities with paper and our tool.

5.2.2 Practitioners—expert modelers

From the solutions, we can infer that practitioners' experience in thinking at different levels of abstraction contributed to the correctness of their metamodels (R2.8). Furthermore, it was interesting to see that almost all of the metamodels were more complete after practitioners had used the wizard, while half of the practitioners said that the wizard does not contribute substantially to their work (R2.9). This could mean that they did not feel the need for assigning types. Unfortunately, our study cannot prove whether the sketches would have become more concrete over time, and if practitioners would then have wanted to assign types or define cardinalities. However, this is the beauty of FlexiSketch: it provides formalization capabilities if needed, but does not force the user to utilize them.

Six out of eleven practitioners did not define all types at the beginning. This is a possible consequence of them not knowing beforehand what notation they are going to use. They concentrated on sketching the problem and came up with notations as needed on the fly. They did not know whether they will reuse certain symbols after drawing them for the first time and therefore delayed the type definition to a later point during the experiment. The fact that our tool allows this is one of its main strengths. The behavior of deferring type definitions contrasts the results of our first study (Sect. 3). When a modeler sketches for herself, she does not need to coordinate or explain to anyone what she is doing. This might foster intentional ambiguity in sketches and could be a possible explanation why type definitions were deferred more in the second study.

One of the questions in our semi-structured interviews concerned possible usage scenarios for our tool. A common answer from the RE practitioners was that they would use the mobile version of FlexiSketch on-site with customers. Sometimes they would also take a photograph of the important machines or surroundings and include it in the sketch. They would then send the sketches consisting of structured information to a coworker. The coworker would further augment and annotate the sketches and send them back for another iteration. Like this, the sketches would evolve into more concrete models over time. As the primary purpose of sketches in this usage scenario is communication, some of the RE practitioners stated that it is not important to define cardinalities.

[12] This could be changed in future tool versions by using placeholders for the relationship types.

In contrast, the practitioners who focus more on software development said that they like the ability to define cardinalities. Cardinalities are needed when models should be processed formally (e.g., for model transformations). In addition, some of them stated that they would like to have even more metamodeling options (such as, e.g., the option to define subtypes of types, and the option to enter complex constraints by using something similar to OCL—the object constraint language).

As long as our tool is used by RE practitioners (our target users), our study suggests that defining cardinalities can remain an optional feature. If, however, our focus would shift to model-driven engineering one day, we should provide additional guidance for defining cardinalities and include further metamodeling options.

5.3 Guidelines for flexible modeling tools

Based on our study findings and earlier research results, we derived a set of guidelines and tips for flexible modeling tools in general. In this subsection, we present the guidelines after briefly explaining what we mean with the term *flexible modeling tool*.

We say that a modeling tool is *flexible* if it supports one or more of the following items. The flexibility of the tool increases with the degree to which it supports these items.

- Accepting undefined model constructs in the modeling canvas and the ability to defer formalization of these model constructs;
- Not imposing a specific workflow (being flexible in the methodology and the order in which artifacts and parts thereof can be created);
- The freedom to use custom modeling languages and notations that are not predefined when the tool is shipped;
- The tool's ability to learn and understand these custom modeling notations and languages;
- Providing compatibility to other modeling tools (in the case that the tool supports metamodeling: import/export of the metamodel).

Flexible modeling tools are of particular importance in creative phases of software projects, in which engineers typically deviate from standard modeling languages [9]. Therefore, we say that a modeling tool is only then truly flexible when new modeling languages can be added or at least existing languages can be augmented. This requires some sort of metamodeling (automatic, by experts, by end-users) to a certain degree (lightweight, full-fledged) at some point in time (before, after, during the creation of models that use the new languages). Correia and Aguiar give an overview of the most familiar types of end-user metamodeling in [7].

Our guidelines are by no means comprehensive; they rather present generalized findings from our studies.

5.3.1 Sketching and Modeling

Think about the effects that tool functionality can have on creativity. Free-form sketching has been shown to be an important enabler for creativity [3,17,37]. Creativity and idea generation can be further actuated by deliberately introducing ambiguity in sketches [17]. In contrast, premature commitment to resolving such ambiguity can stifle creativity. It is therefore important to think about whether certain functionality will foster or discourage premature commitment of users. In the case of FlexiSketch, the drag and drop functionality is an interesting example: because it is the only way to copy elements, users get motivated to assign types to sketched elements early in the workflow, which can lead to an early reduction of ambiguity, distract from the sketching task and therefore can potentially stifle creativity. However, we have experienced that early type assignments can also foster discussions among participants, which can have a positive effect on creativity. We conclude that tool designers should think about which features are really needed in a tool, and how they might affect user behavior in a creative phase.

If there is an inevitable trade-off between flexibility and formalization capabilities, go for flexibility. Practitioners are looking for flexibility. In one of our earlier studies [59], we asked practitioners what we should do if we experience a trade-off between flexibility and formalization capabilities. An overwhelming majority, and especially the practitioners working in the RE field, said that we should go for flexibility.

Provide functionality for increasing the consistency within models. If a flexible modeling tool supports free-form sketching or provides a large number of geometric shapes without explicitly defined meanings, functions for increasing the consistency in a model help to make the models more understandable (also for third parties) and more amenable for subsequent formalization. Once the models become more formal, consistency functionality helps, for example, to obtain a one to one mapping between geometric shapes and meanings. In the case of FlexiSketch, the frequent use of the drag and drop functionality by study participants has noticeably contributed to model consistency.

5.3.2 Collaboration and working style

Support collaboration and involve different stakeholders. Creative sessions (that include brainstorming and coming up with early software design ideas) usually involve more than one person. In the study presented in Sect. 3, participants confirmed that collaboration support is important for tools such as FlexiSketch. We have learned that collaboration mechanisms can motivate stakeholders to actively take part in defining a modeling language. To foster discussions as well as to reach consensus and a higher acceptance of the language, it makes sense to involve not only a language expert in the creation process, but also the persons who actually will use the language. This finding is also confirmed by research of Golra et al. [16] who received very positive feedback from domain experts and clients regarding their inclusion in a flexible modeling process.

Support user awareness. Our collaboration study has shown that it gets difficult to track what is going on as soon as multiple screens are involved, even in case of same-time, same-place collaboration. We have learned that user awareness support must go beyond the visual cues we are used to from online collaboration tools such as Google Docs[13] or virtual whiteboard applications where it often suffices to show the cursor locations of the different users. For example, a shared overview—similar to the one we used in the study—can reduce coordination issues [19]. Furthermore, a tool needs to inform users about metamodeling activities performed by the team members. Participants of our study also stated that a history of user actions, especially in the context of metamodeling, can help to remember and re-discuss certain language decisions by showing who was involved in what parts of the creation process.

Consider the characteristics of the devices used. Which devices to use depends on the type of tool and on the usage scenarios. A tool with finger or pen-based input running on an electronic whiteboard can have a more positive impact on creativity than a tool with mouse and keyboard input running on a desktop computer [15]. In the case of FlexiSketch, some students mentioned that they would have preferred a bigger screen over the mobile tool version, for example, using FlexiSketch Desktop on an electronic whiteboard. In contrast, RE practitioners mentioned a usage scenario where they would use the mobile version of FlexiSketch outside the office when they meet on-site with customers. It is also important to consider the time required to start up a device. Even if a software tool has clear advantages over a classic whiteboard, having to boot a computer and start an application can be reason enough why many people decide to use a classic whiteboard instead.

5.3.3 Metamodeling and formalization

Let the users decide when to perform metamodeling. This guideline directly relates to a part of our definition of flexible modeling tools: do not impose a specific workflow. A tool loses a lot of flexibility if it expects that a model is formalized at a specific point in time, or if it constrains the order of modeling and metamodeling. According to our experience, the fact that users of our approach can choose to perform

[13] docs.google.com.

metamodeling at any time is one of the most important factors for the flexibility of our tool. Approaches that want to provide a flexible interleaving of modeling and metamodeling therefore need to combine both activities in a single tool environment.

Metamodeling becomes easier when users perceive immediate advantages. One of the goals of our FlexiSketch approach is that users willingly perform some lightweight metamodeling activities, and that they do so without really realizing it. We experienced that this works well as long as the users perceive that their actions bring immediate advantages related to their modeling task. For example, the drag and drop functionality of FlexiSketch motivates users to assign types to elements because it allows them to reuse the elements. As a result, the metamodel will already contain types for many model elements when the users finish the modeling task, although they never consciously put effort into performing metamodeling. Flexible modeling approaches should try to find further win–win mechanisms that lead to increased accumulations of metamodel information while providing immediately perceivable advantages to the users.

Distinguish between metamodeling actions and modeling actions. A tool that provides metamodeling and modeling functionality in a single environment should clearly show which menu items, actions, etc., are related to modeling, and which ones are related to metamodeling. This distinction is currently lacking in FlexiSketch. As a consequence, some new users of our tool tend to mix up the context menu items for adding a type to an element and for adding a text box to an element. (However, the users will notice immediately if the element does not appear in the type library and shows a text box instead, or vice versa).

Provide options to show the underlying language definitions. There should be a quick and easy way for the user to check at any time what parts of a model are already formally defined, and what the definitions are. In the case of FlexiSketch, an option in the settings allows users to display the types of defined elements directly on the sketch canvas. In addition, the wizard in the main menu can be used to highlight all untyped elements in a sequential order. However, we only had moderate success with the wizard, and many students did not use it in the study. Thus, we conclude that a tool should use a more easily accessible option that directly shows the underlying language definitions in the working canvas. Such an option could also highlight defined and undefined elements (for example, using different colors).

Provide "metamodeling by example" for novice users. In our second study, many students were not able to define cardinality rules correctly. This confirms studies reporting that end-user metamodeling is hard to achieve (e.g., [46]). We conclude that metamodeling guidance could be improved by a "metamodeling-by-example" mechanism, as studied by Qattous et al. [46]: a tool could show concrete examples which help the users to make correct definitions, or the users could verify their previously entered definitions by looking at concrete examples.

5.4 Threats to validity

In this section, we briefly discuss threats to validity for the two studies we had conducted.

5.4.1 Study 1: simulated workshops

Conclusion validity. The first study was performed with three student groups and three practitioner groups to get qualitative results and an in-depth understanding how groups create ad hoc modeling notations. The small amount of six data points is a threat to conclusion validity. A complementary, quantitative study is necessary to mitigate this threat.

Internal validity. Participants were unfamiliar with the tool, and we gave an introduction to mitigate this threat. Yet, the desire of the participants to explore the new technology could have influenced the collaboration task. For example, it could have fostered simultaneous editing and added to the result that multiple group members participated in defining types. Furthermore, minor usability issues were potential distractions and could have influenced the collaboration task.

When participants know the researchers personally, they might be tempted to give overly positive feedback. To mitigate this threat, we asked students to fill out an anonymous online survey after the lecture. Participants of the two practitioner groups PG2 and PG3 did not know the researcher who performed the study with them, which further mitigates this threat. We did not find a discrepancy between the feedback from PG1 and the other two practitioner groups.

Construct validity. Student groups had to create predefined diagram types. This is a possible threat because it can influence the amount of discussion needed about semantics. (Obviously, student groups talked less about semantics compared to practitioners groups). It can also minimize tool usability issues, since we already knew that these diagrams can be built with our tool. However, some of the results, such as the lack of micro-coordination in student groups, are independent of a particular modeling notation and therefore not affected by this threat. Furthermore, students had to solve a task that was constructed by us, while practitioners could work on a real-world task from their company. The participants might thus have had different levels of intrinsic motivation.

We told the groups that all elements should have types assigned at the end. This could have influenced their typing behavior and therefore might have contributed to the result of consistent notations. However, we wanted to make sure that the groups define enough types (or define types at all), such

that we can draw conclusions about their language definition behavior.

External validity. The limited number of students and practitioners who were involved in our evaluation activities, as well as the limited geographical distribution (Switzerland and Austria), is a known threat (convenience sampling according to proximity). However, we involved both novice and expert modelers with different backgrounds and skills. During the 20 min sessions, we identified collaboration patterns that confirm the usefulness of our FlexiSketch approach for early requirements modeling. Whether the results can be generalized to longer sessions has yet to be verified. The behavior of modelers might differ over time. Furthermore, the results are valid for same-time, same-place collaboration. Results for distributed teams might differ from our observations.

5.4.2 Study 2: quantitative experiment

Conclusion validity. In our second study, the limited number of eleven practitioners involved could be seen as threat. However, we will continue to perform reality checks with practitioners, gathering additional data for the statements made in this article. Also, results from a previous experiment [59] do not contradict the latest findings. The student experiment was of a quantitative nature. It was performed at two Swiss universities, and thus, the locality could be seen as threat.

Internal validity. Participants stated they were familiar with tablets or touch devices in general. However, they were using FlexiSketch for the first time. We included a tutorial at the beginning of the experiment to mitigate the threat to internal validity. However, we expect that involving trained FlexiSketch users would have led to better results. Furthermore, the student experiment took place as part of a lecture, which could have influenced students' motivation and therefore the quality of the results. Also, differences between the quality of class diagram and use case diagram solutions (including the metamodel) could be an effect of differences between the students of the two universities. All students were at the beginning of learning to model UML. At the time of the experiment, there was no UML model type that was known by students from both universities. Thus, we could not mix the students from the two universities.

Regarding the differences between FlexiSketch solutions and paper solutions, it has to be noted that the different versions of the handouts could have affected the outcome. For example, presenting the modeling task, the type definition task, and the cardinality definition task as individual parts in the handout for the non-FlexiSketch version could have been too much of a help for those students. We decided to do this in order to have an equivalent help compared to the tool's wizard. However, many students who performed their task with the tool did not consult the wizard. This could explain why paper solutions outperformed tool solutions. Furthermore, the tool's restriction of permitting cardinality definitions only for relations that have already been fully typed was perceived as an usability problem by some students.

Another issue is that we used standard modeling notations within the student experiment. While our approach is rather meant to be used with custom languages, this allowed us to precisely measure completeness and correctness against a given ground truth and therefore measure the metamodeling capabilities of novice modelers when supported by our tool.

Construct validity. When planning the experiment, we tried to avoid several threats to construct validity. For example, we conducted pilot tests with graduate students to verify that our handouts are understandable and that participants have enough time for the main modeling task without suffering from time pressure. In the student experiment, students were randomly assigned to either work on the tasks with a tablet, or to be part of the control group and work on paper. This allowed us to compare tool solutions with solutions on paper and therefore measure whether our tool itself biased the results.

Another possible threat is the absence of an objective ground truth against which we could have evaluated the results from practitioners. Metamodel completeness could be deduced from the model sketches, but for assessing metamodel correctness, we partly had to rely on practitioners' opinions since the used languages only existed in practitioners' minds.

In order to be able to compare student and practitioner solutions in terms of quality, we tried to give the same instructions in both experiments (apart from having a predetermined model type and task in the student experiment). For example, we mentioned the existence and functionality of the tool wizard in the same way. We only forced practitioners to use the wizard once they were done with the task (such that we could compare their prewizard results with student results). However, the students received their task as written instructions, while we told the instructions to practitioners orally.

External validity. In order to reduce threats, we evaluated FlexiSketch with both experienced modelers and modeling novices. We included different diagram types, both custom and standardized notations, to increase the generalizability of the experiment results.

6 Related work

6.1 Sketching in SE

Sketching is an important method to foster creativity and discuss design ideas [3,17,37]. Researchers have identified the

need for sketching in software engineering to support creativity and idea generation a long time ago [12,53]. Various studies investigate the reasons for the popularity of whiteboards and sketching in SE [4,15,37,55].

As a result, there exist many approaches that augment formal modeling tools with sketch recognition, for example, SUMLOW [3], Tahuti [22], SketchREAD [1], and Scribble [50]. A detailed overview is provided by Johnson et al. [27]. Furthermore, many sketch recognition-based approaches focus on user interface design, e.g., [8,32]. These approaches have in common that they start from predefined modeling notations and then provide a sketch interface capable of recognizing these notations. Therefore, the tools cannot interpret any drawings which do not conform to the given notations. In contrast, our approach starts on the informal side, mimicking paper, and permits the use of arbitrary notations.

On the informal end of the spectrum, tools such as Calico [38] and Sketch for Eclipse [48] support informal sketching with the possibility to structure the information, but they do not provide further formalization capabilities. AugIR [29] is a sketching tool running on multiple electronic whiteboards and enables users to add multiple annotations to sketched elements. In contrast to our work, their goal is not to distill metamodel information for a formalization of the sketches, but they use the annotations to automatically link related concepts found in other sketches to the annotated elements. This allows users to easily navigate between related sketches and artifacts. BITKit [42] is an approach that bridges the gap between office tools and modeling tools. It allows users to add tags to geometrical shapes and connections for categorizing them. Tagged connections can require shapes with certain tags as endpoints. However, BITKit does not contain sketching. While the authors discuss briefly on a theoretical level how the diagrams can be formalized and how an underlying metamodel can be created, the prototype does not contain such functionality. InkKit [45] allows to add additional diagram types including recognition and export support, but uses DLL plug-ins for this purpose.

6.2 End-user metamodeling

Metamodeling tools such as MetaBuilder [13] or MetaEdit+ [28] provide graphical metamodel editors for the creation of modeling languages and editors. DiaMeta [41] and MaramaSketch [18] allow to develop diagram editors which support freehand editing. However, with these tools, users cannot create custom modeling languages by example, but must define them beforehand in the metamodeling tool. Once the editor tools are compiled, languages cannot be extended directly in these editors. In contrast, our flexible metamodeling approach uses metamodeling by example. This also allows for an iterative metamodel creation process where business stakeholders can draw models before the metamodel is complete. Similarly, the Electronic Cocktail Napkin [17] provides a sketching interface and allows users to define constructs later on in order to resolve ambiguity and vagueness. However, the tool is not tailored to the SE domain and requires scripting or programming knowledge to create metadescriptions.

The operation of metamodeling tools usually requires good metamodeling knowledge. In contrast, there are not many approaches for end-user metamodeling (i.e., non-metamodeling experts). One reason is that, from a metamodeling perspective, it was long believed that metamodeling should only be done by metamodeling experts [30]. Indeed, it has been shown that end-user metamodeling is hard to achieve (e.g., see [46]). In contrast, we concentrate on lightweight metamodeling (or just enough metamodeling) by example for creating ad hoc notations in an end-user friendly way (e.g., for requirements engineers and domain experts). Our approach considers the results of Qattos et al. [46], who report on experiments showing that metamodeling by example (seeing concrete graphical examples) results in better metamodels than a wizard-based method. Furthermore, Cho et al. [5] discuss challenges of metamodeling by example, and Kuhrmann [31] argues for the necessity of user assistance for DSML creation. One of the challenges lies in the coevolution of models and metamodels. Coevolution issues are also discussed in [6,10,47,57]. Since our main goal is to formalize model sketches, we have neglected the coevolution topic so far by having one metamodel per model sketch, and always updating the metamodel according to the sketch. But coevolution should be a topic for future work. Regarding coevolution, there is a growing number of frameworks using the dynamic typing capabilities of JavaScript to support flexible modeling. For example, FlexiMeta [23] and the JavaScript Modeling Framework (JSMF) [52] allow a user to create a model that is independent of any metamodel during an exploration phase. In later phases, a metamodel can be created, and the user can apply modifications to align the model with the metamodel. Both approaches mentioned are not graphical modeling tools, but frameworks that can be implemented in the latter. JSMF currently allows to create models and metamodels by writing JavaScript code. In the case of FlexiMeta, it will depend on the implementing tool (and is not clear yet) how much metamodeling the user will be able to perform, and what the usability of the approach will be.

Publications about the evaluation of end-user metamodeling are still scarce. For example, Gabrysiak et al. [14] present different approaches to create metamodels before or after creating models in their position paper, but do not provide an evaluation. Volz and Jablonski [56] propose an approach that allows the step-wise formalization of sketched models. Cuadrado et al. [47] and López-Fernández et al. [36]

propose bottom-up metamodeling, similar to our approach, but they use separate tools for the modeling and metamodeling activities. No user evaluation is provided in any of these papers. Wouters [58] proposes a notation-driven approach to create a metamodel, which is supposed to help in creating notations that match domain experts' expectations. The work is evaluated in an industrial case study and an empirical study. However, the approach uses a graphical grammar instead of free sketching. Another thread of research investigates data mining technology for automatic creation of metamodels from a large set of given example models, e.g., MARS [25,40]. In contrast, FlexiSketch aims at building metamodels for a small set of sketches or individual sketches, so approaches requiring large data sets are not applicable in our case. Similarly, the Muddles flexible modeling approach [63] creates a metamodel from a single model: it lets a user create a model and annotate it with types by using a GraphML compliant tool such as yEd.[14] Afterward, another tool automatically generates a metamodel, which includes the inference of cardinality rules and missing types. In contrast to FlexiSketch, there is no interleaving of modeling and metamodeling. A very flexible interleaving of these activities is provided by the Free Modeling Environment [16]. It enables a user to work on models and metamodels side by side at the same time. The models are loosely bound; the user can decide whether a change in a (meta)model should be propagated to the other (meta)models or not. The approach focuses on representation rather than on further metamodeling definitions such as cardinalities and constraints. However, by displaying the metamodels directly to the user, the tool is meant to be used by language experts when they collaborate with client stakeholders. In contrast, FlexiSketch lets the user focus on modeling and creates the metamodel behind the scenes, which is always bound to the model.

To summarize, FlexiSketch does not compete with other existing flexible modeling tools, but it complements them. This is also shown in Table 10: the combination of a sketching environment (row 3) with the possibility to interleave modeling and metamodeling (rows 6 and 7) in a single environment (rows 10 and 11) by providing a lightweight form of end-user metamodeling is a unique characteristic of our tool. Other tools do either not provide sketching, require significant metamodeling knowledge (or have very limited metamodeling functionality), or do not support the interleaving of modeling and metamodeling in a single environment. Thus, FlexiSketch is a new kind of tool that fills a gap between flexible sketching tools with limited formalization capabilities and sophisticated metamodeling tools for language experts.

[14] https://www.yworks.com/products/yed.

6.3 Collaborative design and language creation

In requirements and software engineering, collaboration is often researched in the context of design [2,35], and user interface creation [26,35,49]. Some researchers focus on understanding the behavior and low-level collaboration patterns of participants when working with physical media, e.g., [20,54], which resulted in design guidelines for software tools that support collaborative work [19–21,54]. Today, there are many software tools that support collaborative sketching and design work (e.g., Calico [38], The NiCE Discussion Room [21]). Collaboro [24] is a collaborative metamodeling tool that provides explicit collaboration functionality focusing on the collaboration process itself, such as the explicit support for change requests, discussions, community votes, keeping track of possible solutions, comments and decisions, and more. Settings with such digital tools have the potential to change the way how engineers and designers work and collaborate. Users might show different collaboration behaviors if they work with digital tools instead of physical media, because, e.g., workspace awareness can be different. Therefore, when introducing a new software tool, it makes sense to study the influence of this tool on collaboration and sketching behavior. Examples of such studies can be found in, e.g., [21,34,38,39].

While we also looked at collaborative sketching behavior when using FlexiSketch, the main focus of our first study was to investigate how requirements engineers collaboratively design, agree on, and define notations. Related work on this subject is still scarce. Dekel and Herbsleb [9] performed an observational study to find out what kind of notations are used in object-oriented design, and how they evolve during sessions. Ossher et al. [43] investigated notations used in software design sessions to conclude whether their flexible modeling approach can provide appropriate support. Both works used physical media in the studies. In contrast, our study investigates how non-expert metamodelers choose and define notations when using a flexible software tool.

7 Conclusions

As a part of our flexible modeling approach, we developed the FlexiSketch tool. The multi-screen, node-and-edge diagram sketching tool allows users to define custom notations on the fly by assigning types to elements and specifying cardinality rules. In this work, we presented two studies about lightweight metamodeling using our tool: a qualitative study about how requirements engineers sketch and define ad hoc notations collaboratively, and a quantitative study about how well modeling novices can handle the metamodeling mechanisms. The qualitative study indicates that the tool fosters interleaving of sketching and type-defining activities, and

Table 10 Comparison of flexible modeling tools regarding their inclusion of sketching, modeling and metamodeling, and the type of tool

		Classic whiteboard	Calico [38]	SUMLOW [3]	Electronic Cocktail Napkin [17]	BITKit [42]	Free Modeling Environment [16]	Muddles [63]	Cuadrado et al. [47]	FlexiSketch	FlexiMeta [23]	JSMF [52]	MetaEdit+ [28]	Marama [18]
Modeling (by)	Scripting/programming											✔		
	Drag&drop		✔			✔	✔	✔	✔	✔			✔	✔
	Free sketching	✔	✔	✔	✔			( )[1]	( )[1]	✔				✔
	Sketch recognition			✔	✔			( )[1]	( )[1]	✔				✔
	Formalization			✔	✔	( )[2]	✔	✔	✔	✔	✔	✔	✔	✔
Metamodeling	Custom notations (concrete syntax)				✔	✔	✔	(✔)[3]	(✔)[3]	✔			✔	✔
	Custom languages (abstract syntax)					(✔)[2]	(✔)[2]	✔	✔	✔	✔	✔	✔	✔
	Custom semantics											( )[2]	✔	(✔)[2]
	Detail of metamodeling (0-3)	0	0	0	1	1	1	2	2	2	2	3	3	3
Interplay modeling/ metamodeling	Can be done in same environment/ tool	–[4]	–[4]	–[4]	✔	✔	✔			✔	✔	✔		
	Modeling possible before metamodeling	–[4]	–[4]	–[4]	✔	✔	✔	✔	✔	✔	✔		✔	✔
Tool type	Tool category (1=prototype, 2= framework, 3=production)	3	3	1	1	1	1	1	1	1	2	2	3	3
	Input method (mk=mouse&keyboard, p=pen-based, f=finger, na= no info)	p	f, p, mk	p	p	mk	na	na	na	f, p, mk	na	na	mk	p, mk
	Hardware (w=e-whiteboard, t=tablet, d=desktop, na= no info)		w, t, d	w	t, d	na	na	na	na	w, t, d	na	na	d	na
Target users	No programming/scripting required	✔	✔	✔		✔	✔	✔		✔	(✔)[5]		(✔)[6]	(✔)[6]
	Non-metamodeling experts can create initial metamodel information		✔	✔	✔	✔		✔	( )[7]	✔	(✔)[5]			( )[7]
	Explicit collaboration functionality		✔		✔					✔			(✔)[2]	✔

For each group of flexible modeling approaches discussed in Sects. 6.1 and 6.2, one or two representative tools have been chosen and included in the comparison table

[1] The approach itself does not support sketching, but uses third-party modeling tools, some of which might include sketching

[2] The support of the feature is noticeably more limited compared to the approaches that have a checkmark in the same row

[3] Due to the multi-tool workflow, end-users can only define custom notations when creating new languages

[4] The criterion cannot be applied to these tools because they do not support any metamodeling features

[5] The framework itself requires some knowledge, but the demo application that integrates the framework does not

[6] Scripting/programming is required for the mapping of concepts to source code

[7] Some knowledge is required

motivates all members of a group to perform both activities. Users managed to define consistent notations for their sketches collaboratively and reached a common understanding of the respective notations. The quantitative study shows that novice modelers with no metamodeling knowledge can define types on a metalevel, but have trouble in correctly defining cardinality rules and sometimes think on the model level instead of the metamodel level.

Results from the first study also suggest that having additional awareness features in the tool (for knowing what the other users are doing) would be beneficial. Furthermore, the second study shows that active guidance by the tool is needed if novice modelers shall be able to provide correct cardinality rules. One option could be to show concrete examples as done in [46]. We also saw that some users are only willing to provide metainformation if they receive immediate benefits, e.g., by reusing types via drag and drop. While not all practitioners would want to formalize the sketches created with our tool, they understand the importance of providing metainformation to allow the formalization of model sketches such that they can be exported and reused in other tools. In our future work, we plan to improve FlexiSketch according to these results. This includes the integration of additional user awareness features for the collaborative version, tool guidance for cardinality definitions, and usability improvements. We also plan to perform longitudinal evaluations in real-world software projects, and investigate how sketches made with our tool are reused and changed during the projects. This will allow us to gather feedback about the quality of sketches and metamodels from people who will actually have to reuse these artifacts as a part of their work.

Acknowledgements We would like to thank the journal reviewers and André van der Hoek for their very valuable feedback and comments on earlier versions of this article.

References

1. Alvarado, C., Davis, R.: SketchREAD: a multi-domain sketch recognition engine. In: 17th Annual ACM Symposium on User Interface Software and Technology (UIST 2004), pp. 23–32. ACM (2004)
2. Branham, S., Golovchinsky, G., Carter, S., Biehl, J.T.: Let's go from the whiteboard: supporting transitions in work through whiteboard capture and reuse. In: 28th International Conference on Human Factors in Computing Systems (CHI 2010), pp. 75–84. ACM (2010)
3. Chen, Q., Grundy, J., Hosking, J.: SUMLOW: early design-stage sketching of UML diagrams on an e-whiteboard. Softw. Pract. Exp. **38**(9), 961–994 (2008)
4. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J.: Let's go to the whiteboard: how and why software developers use drawings. In: 25th International Conference on Human Factors in Computing Systems (CHI 2007), pp. 557–566. ACM (2007)
5. Cho, H., Gray, J., Syriani, E.: Creating visual domain-specific modeling languages from end-user demonstration. In: 2012 ICSE Workshop on Modeling in Software Engineering (MISE), pp. 22–28 (2012)
6. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2008), pp. 222–231. IEEE Computer Society (2008)
7. Correia, F.F., Aguiar, A.: Patterns of flexible modeling tools. In: 20th Conference on Pattern Languages of Programs (PLoP 2013). Article No. 9. The Hillside Group (2013)
8. Coyette, A., Schimke, S., Vanderdonckt, J., Vielhauer, C.: Trainable sketch recognizer for graphical user interface design. In: Baranauskas, C., Palanque, P., Abascal, J., Barbosa, S.D.J. (eds.) Human-Computer Interaction—INTERACT 2007, LNCS, vol. 4662, pp. 124–135. Springer, Berlin (2007)
9. Dekel, U., Herbsleb, J.D.: Notation and representation in collaborative object-oriented design: An observational study. SIGPLAN Not. **42**(10), 261–280 (2007)
10. Di Ruscio, D., Iovino, L., Pierantonio, A.: Managing the coupled evolution of metamodels and textual concrete syntax specifications. In: 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2013), pp. 114–121 (2013)
11. Do, Q., Cook, S., Campbell, P., Scott, W., Robinson, K., Power, W., Tramoundanis, D.: Requirements for a metamodel to facilitate knowledge sharing between project stakeholders. Procedia Comput. Sci. **8**, 285–292 (2012)
12. Ellis, T.O., Heafner, J.F., Sibley, W.L.: The GRAIL project: an experiment in man-machine communications. Tech. Rep. RAND Memorandum RM-5999-ARPA, RAND Corporation (1969)
13. Ferguson, I., Hunter, A., Hardy, C.J.: Metabuilder: the diagrammer's diagrammer. In: Theory and Application of Diagrams. LNCS, vol. 1889, pp. 407–421. Springer (2000)
14. Gabrysiak, G., Giese, H., Lüders, A., Seibel, A.: How can metamodels be used flexibly? In: ICSE Workshop on Flexible Modeling Tools. http://www.researchgate.net/publication/235675453_How_Can_Metamodels_Be_Used_Flexibly (2011). Accessed 7 Apr 2016
15. Goel, V.: Sketches of Thought. The MIT Press, Cambridge (1995)
16. Golra, F.R., Beugnard, A., Dagnat, F., Guérin, S., Guychard, C.: Using free modeling as an agile method for developing domain specific modeling languages. In: 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), pp. 24–34. ACM (2016)
17. Gross, M.D., Do, E.Y.L.: Ambiguous intentions: a paper-like interface for creative design. In: 9th Annual ACM Symposium on User Interface Software and Technology (UIST 1996), pp. 183–192. ACM (1996)
18. Grundy, J., Hosking, J.: Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In: 29th International Conference on Software Engineering (ICSE 2007), pp. 282–291. IEEE Computer Society (2007)
19. Gutwin, C., Greenberg, S.: Design for individuals, design for groups: tradeoffs between power and workspace awareness. In: 1998 ACM Conference on Computer Supported Cooperative Work (CSCW), pp. 207–216. ACM (1998)
20. Gutwin, C., Greenberg, S.: The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces. In: 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000), pp. 98–103. IEEE (2000)
21. Haller, M., Leitner, J., Seifried, T., Wallace, J.R., Scott, S.D., Richter, C., Brandl, P., Gokcezade, A., Hunter, S.: The NiCE Discussion Room: integrating paper and digital media to support co-located group meetings. In: 28th International Conference on Human Factors in Computing Systems (CHI 2010), pp. 609–618. ACM (2010)
22. Hammond, T., Davis, R.: Tahuti: a geometrical sketch recognition system for UML class diagrams. In: ACM SIGGRAPH 2006 Courses, Article No. 25, ACM (2006)
23. Hili, N.: A metamodeling framework for promoting flexibility and creativity over strict model conformance. In: 2nd Workshop on Flexible Model Driven Engineering Co-Located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), pp. 2–11. CEUR-WS.org (2016)

24. Izquierdo, J.L.C., Cabot, J.: Collaboro: a collaborative (meta) modeling tool. Peer J. Comput. Sci. **2**, e84 (2016)
25. Javed, F., Mernik, M., Gray, J., Bryant, B.R.: Mars: a metamodel recovery system using grammar inference. Inf. Softw. Technol. **50**(9–10), 948–968 (2008)
26. Johansson, M., Arvola, M.: A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings. In: 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it (BCS-HCI 2007), pp. 177–184. British Computer Society (2007)
27. Johnson, G., Gross, M.D., Hong, J., Yi-Luen Do, E.: Computational support for sketching in design: a review. Found. Trends Hum.-Comput. Interact. **2**(1), 1–93 (2009)
28. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment. In: Constantopoulos, P., Mylopoulos, J., Vassiliou, Y. (eds.) Advanced Information Systems Engineering, LNCS, vol. 1080, pp. 1–21. Springer, Berlin (1996)
29. Kleffmann, M., Hesenius, M., Gruhn, V.: Connecting UI and business processes in a collaborative sketching environment. In: 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2015), pp. 200–205. ACM (2015)
30. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Professional, Boston (2008)
31. Kuhrmann, M.: User assistance during domain-specific language design. In: ICSE Workshop on Flexible Modeling Tools. https://www4.in.tum.de/publ/papers/kuhrmann2011a.pdf (2011). Accessed 7 Apr 2016
32. Landay, J.A., Myers, B.A.: Sketching interfaces: toward more human interface design. Computer **34**(3), 56–64 (2001)
33. Lee, J.S., Tatar, D., Harrison, S.: Micro-coordination: because we did not already learn everything we need to know about working with others in kindergarten. In: 2012 ACM Conference on Computer Supported Cooperative Work (CSCW), pp. 1135–1144. ACM (2012)
34. Lee, L.C., Wei, W.J.: Comparison study on sketching behaviors with and without interactive pen displays. In: 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2007), pp. 567–574. Springer, Berlin (2008)
35. Loksa, D., Mangano, N., LaToza, T.D., van der Hoek, A.: Enabling a classroom design studio with a collaborative sketch design tool. In: 35th International Conference on Software Engineering (ICSE 2013), pp. 1073–1082. IEEE (2013)
36. López-Fernández, J.J., Cuadrado, J.S., Guerra, E., Lara, J.: Example-driven meta-model development. Softw. Syst. Model. **14**(4), 1323–1347 (2015)
37. Van der Lugt, R.: Functions of sketching in design idea generation meetings. In: 4th Conference on Creativity & Cognition (C&C 2002), pp. 72–79. ACM (2002)
38. Mangano, N., Baker, A., Dempsey, M., Navarro, E., van der Hoek, A.: Software design sketching with Calico. In: 2010 IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 23–32. ACM (2010)
39. Mangano, N., LaToza, T.D., Petre, M., van der Hoek, A.: Supporting informal design with interactive whiteboards. In: 32nd International Conference on Human Factors in Computing Systems (CHI 2014), pp. 331–340. ACM (2014)
40. Mernik, M., Hrncic, D., Bryant, B.R., Sprague, A.P., Gray, J., Liu, Q., Javed, F.: Grammar inference algorithms and applications in software engineering. In: 22th International Symposium on Information, Communication and Automation Technologies (ICAT 2009), pp. 1–7. IEEE (2009)
41. Minas, M.: Generating meta-model-based freehand editors. In: 3rd International Workshop on Graph Based Tools (GraBaTs 2006), Satellite event of the 3rd International Conference on Graph Transformation, Electronic Communications of the EASST, vol. 1 (2006)
42. Ossher, H., Bellamy, R., Simmonds, I., Amid, D., Anaby-Tavor, A., Callery, M., Desmond, M., de Vries, J., Fisher, A., Krasikov, S.: Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. In: 2010 ACM International Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA), pp. 848–864. ACM (2010)
43. Ossher, H., John, B., Desmond, M., Bellamy, R.: Are flexible modeling tools applicable to software design discussions? IBM Tech. Rep. RC24949 (W1002-054) (2010)
44. Plimmer, B., Apperley, M.: INTERACTING with sketched interface designs: an evaluation study. In: 2004 CHI Extended Abstracts on Human Factors in Computing Systems (CHI EA), pp. 1337–1340. ACM (2004)
45. Plimmer, B., Freeman, I.: A toolkit approach to sketched diagram recognition. In: 21st British HCI Group Annual Conference on People and Computers (BCS-HCI 2007), pp. 205–213. British Computer Society (2007)
46. Qattous, H., Gray, P., Welland, R.: An empirical study of specification by example in a software engineering tool. In: 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Article No. 16. ACM (2010)
47. Sánchez-Cuadrado, J., De Lara, J., Guerra, E.: Bottom-up meta-modelling: An interactive approach. In: 15th International Conference on Model-Driven Engineering Languages and Systems (MODELS 2012), pp. 3–19. Springer, Berlin (2012)
48. Sangiorgi, U.B., Barbosa, S.D.: Sketch: Modeling using freehand drawing in eclipse graphical editors. In: ICSE Workshop on Flexible Modeling Tools. http://www.academia.edu/369660/SKETCH_Modeling_Using_Freehand_Drawing_In_Eclipse_Graphical_Editors (2010). Accessed 7 Apr 2016
49. Sangiorgi, U.B., Beuvens, F., Vanderdonckt, J.: User interface design by collaborative sketching. In: 2012 ACM Conference on Designing Interactive Systems (DIS), pp. 378–387. ACM (2012)
50. Scharf, A., Amma, T.: Dynamic injection of sketching features into GEF based diagram editors. In: 35th International Conference on Software Engineering (ICSE 2013), pp. 822–831. IEEE Press (2013)
51. Shih, P.C., Nguyen, D.H., Hirano, S.H., Redmiles, D.F., Hayes, G.R.: GroupMind: Supporting idea generation through a collaborative mind-mapping tool. In: 2009 ACM International Conference on Supporting Group Work (GROUP), pp. 139–148. ACM (2009)
52. Sottet, J., Biri, N.: JSMF: a javascript flexible modelling framework. In: 2nd Workshop on Flexible Model Driven Engineering co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), pp. 42–51. CEUR-WS.org (2016)
53. Sutherland, I.E.: Sketchpad: a man-machine graphical communication system. Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering (1963)
54. Tang, J.C.: Findings from observational studies of collaborative work. Int. J. Man Mach. Stud. **34**(2), 143–160 (1991)
55. Tversky, B.: What do sketches say about thinking? In: 2002 AAAI Spring Symposium on sketch understanding. AAAI Press (2002)
56. Volz, B., Zeising, M., Jablonski, S.: The open meta modeling environment. In: ICSE Workshop on Flexible Modeling Tools. http://www.researchgate.net/publication/266871141_The_Open_Meta_Modeling_Environment (2011). Accessed 7 Apr 2016
57. Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: 21th European Conference on Object-Oriented Programming (ECOOP 2007), LNCS, vol. 4609, pp. 600–624. Springer, Berlin (2007)
58. Wouters, L.: Towards the notation-driven development of DSMLs. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.)

Model-Driven Engineering Languages and Systems, LNCS, vol. 8107, pp. 522–537. Springer, Berlin (2013)
59. Wüest, D., Seyff, N., Glinz, M.: FlexiSketch: a mobile sketching tool for software modeling. In: 4th International Conference on Mobile Computing, Applications and Services (MobiCASE 2012), pp. 225–244. Springer, Berlin (2013)
60. Wüest, D., Seyff, N., Glinz, M.: Semi-automatic generation of metamodels from model sketches. In: 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013), pp. 664–669. IEEE (2013)
61. Wüest, D., Seyff, N., Glinz, M.: FLEXISKETCH TEAM: Collaborative sketching and notation creation on the fly. In: 37th International Conference on Software Engineering (ICSE 2015), pp. 685–688. IEEE Press (2015)
62. Wüest, D., Seyff, N., Glinz, M.: Sketching and notation creation with FlexiSketch Team: Evaluating a new means for collaborative requirements elicitation. In: 23rd IEEE International Requirements Engineering Conference (RE 2015), pp. 186–195. IEEE (2015)
63. Zolotas, A., Matragkas, N.D., Devlin, S., Kolovos, D.S., Paige, R.F.: Type inference in flexible model-driven engineering. In: Modelling Foundations and Applications - 11th European Conference, (ECMFA 2015), Held as Part of STAF 2015, pp. 75–91. Springer (2015)

Dustin Wüest obtained a Ph.D. degree in Computer Science from the University of Zurich in 2016. He currently holds postdoctoral research positions in the Centre for Requirements Engineering at the FHNW University of Applied Sciences and Arts Northwestern Switzerland, and in the Requirements Engineering Research Group at the University of Zurich. His interests include software and requirements modeling, human computer interaction, and computer graphics.

Norbert Seyff is a professor at the FHNW University of Applied Sciences and Arts Northwestern Switzerland and a senior research associate at the University of Zurich. Norbert holds a doctorate degree in Computer Science from Johannes Kepler University Linz, Austria. His current research focuses on requirements and software engineering. Norbert has a particular interest in empowering and supporting end-users' participation in software development and evolution.

Martin Glinz was a professor of Informatics at the Department of Informatics of the University of Zurich (UZH) from 1993 until July 2017. From 2007 to 2016, he also was the department head. He is now a full professor emeritus, but continues to be active in research and teaching. His interests include requirements and software engineering—in particular modeling, validation, quality, and evolution. He received his Dr. rer. nat. in Computer Science from RWTH Aachen University. Before joining the University of Zurich, he worked in industry for ten years.